Juha Keponen

Hajautettuun tietojärjestelmään perustuva terveyskioski

Sähkötekniikan korkeakoulu

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 5.8.2016.

Työn valvoja:

Prof. Raimo Sepponen

Työn ohjaaja:

TkL Matti Linnavuo

Aalto-yliopisto
Sähkötekniikan korkeakoulu

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Juha Keponen

Työn nimi: Hajautettuun tietojärjestelmään perustuva terveyskioski

Päivämäärä: 5.8.2016 Kieli: Suomi Sivumäärä:6+54

Sähkötekniikan ja automaation laitos

Professuuri: Bioniikka Koodi: S-66

Valvoja: Prof. Raimo Sepponen

Ohjaaja: TkL Matti Linnavuo

Tämän diplomityön on tarkoitus jatkokehittää Juha-Matti Santeron kehittämää terveystuolia ja siihen kuuluvaa käyttöliittymää.
Diplomityön tarkoitus on asentaa ihmisen fysiologisia ominaisuuksia mittaava laitteisto terveyskioskiin käyttämällä niin valmiiksi tilattua laitteistoa kuin suunnittelemalla osa laitteistosta itse. Kioski on tilattu Aalto-yliopiston taiteen ja suunnittelun korkeakoulusta projektityönä.
Teoriaosassa kerrotaan ensin perustietoa mittauksista, minkä jälkeen kerrotaan itse laitteistosta ja sen toiminnasta. Lopuksi annetaan ideoita jatkokehittelyä varten.

Avainsanat: terveystuoli, fysiologia, terveyskioski, hajautettu järjestelmä

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Juha Keponen

Title: Health kiosk based on a distributed system

Date: 5.8.2016 Language: Finnish Number of pages:6+54

Department of Electrical Engineering and Automation

Professorship: Bionics Code: S-66

Supervisor: Prof. Raimo Sepponen

Instructor: Lic.Sc. (Tech.) Matti Linnavuo

To further develop and improve distributed measurement system to be used in a health kiosk. This platform was made by students from Aalto University School of Arts, Design and Architecture.
The purpose of this health kiosk is to serve students and staff alike to be able to conduct basic health measurements independently in an easy accessible public place.
The theory behind the measurements is firstly discussed in this work. Then the functionality of the system as well as the programming solutions used in this work are laid out. Finally, further development ideas are discussed for future versions.

Keywords: health chair, health kiosk, physiology, distributed system

Esipuhe

Haluan kiittää erikoistutkija Matti Linnavuota loistavasta ohjauksesta sekä perhettäni ja mummiani tukemisestani tämän työn kirjoittamisessa. Lisäksi kiitos siskolleni ja äidilleni työn oikolukemisesta ja ideoista.

Otaniemi, 5. elokuuta 2016

Juha M. Keponen

Sisältö

Symbolit ja lyhenteet

Lyhenteet

ADC	Analog-to-Digital Converter, analogi-digitaalimuunnin
AV node	Atrioventricular node, eteis-kammiosolmuke
BMI	Body Mass Index, painoindeksi
ECG	Electrocardiography, sydämen sähkökäyrä
ECW	Extra-cellular water, solun ulkopuolinen vesi
FFM	Fat-free mass, rasvaton massa
ICW	Intra-cellular water, solun sisäinen vesi
SA node	Sinoatrial node, sinussolmuke
SpO2	Veren happisaturaatio
TBW	Total-body water, kehon kokonaisvesi

1 Johdanto

Tulevaisuudessa suurempi osa väestöstä tulee olemaan vanhempia [29], jolloin terveyteen tullaan kiinnittämään yhä enemmän huomiota. Erilaisille itsenäisille digitaalisille järjestelmille, kuten teryeystuoleille, älyrannekelloille, älypuhelimille, älysängyille ja erilaisille terveyskioskeille tulee olemassa potentiaalisesti suuret ja kasvavat markkinat [24]. Käyttäjän eri fysiologisia ominaisuuksia mittaavia terveyskioskeja voisi olla esimerkiksi kauppakeskuksissa, lentokentillä tai muilla yleisillä paikoilla. Kioskien käytettävyyden pitäisi olla niin yksinkertaista, että asiakkaat osaisivat käyttää niitä itse ilman apua, tällöin kioskit eivät sitoisi terveydenhoitohenkilökuntaa vaan he voisivat keskittyä hoitoa vaativiin vaivoihin. Käytön jälkeen asiakas voisi tallentaa tiedot johonkin tietokantaan lääkärin nähtäväksi, tulostaa tiedot itselleen tai vain poistaa ne niin halutessaan.

Terveystuolin ja terveyskioskin ero on yleensä siinä, että terveyskioski tarkoittaa pistettä, johon käyttäjä voi mennä yksin mittaamaan terveystietojaan. Kuvassa 1, on nähtävissä Pursuant Health -nimisen yrityksen teryeyskioski. Pelkkällä terveystuolilla taas tarkoitetaan yleensä esimerkiksi sairaaloissa olevia tuoleja, joita henkilökunta käyttää. Tällaiset monikäyttöiset asemat ovat käytännöllisiä esimerkiksi iäkkäiden potilaiden terveydentilan arvioinnissa, kun potilaasta voidaan mitata yhdellä kerralla kaikki tarvittava. Tämä on myös potilaiden etu, sillä heidän ei tarvitse stressata ja odottaa pääsyä eri mittauksiin.

Kuva 1: Esimerkki terveyskioskista

Kirjallisuudessa terveyskioskilla voidaan myös tarkoittaa pistettä, jossa kysytään käyttäjältä tietoa esimerkiksi hänen syömistottumuksistaan ja sen perusteella annetaan tietoa asiakkaan terveydestä tai pyritään vähentämään riskikäyttäytymistä, kuten alkoholin liikakäyttöä, tupakointia tai lääkkeiden ja huumeiden väärinkäyttöä. Tässä työssä ei paneuduta tällaisiin kioskeihin, mutta niiden käytöstä terveystiedon levittämisessä on positiivisia kokemuksia [17].

Suomen itsenäisyyden juhlarahasto Sitralla oli käynnissä vuosina 2009–2012 käynnissä kokeilu, jossa Suomen kunnat testasivat terveystuolia. Ensimmäinen terveys-

kioski avattiin Ylöjärven kauppakeskukseen vuonna 2009. Sen jälkeen niitä on avattu eri puolilla Suomea ja vuonna 2013 niitä oli toiminnassa noin 30 kappaletta [26]. Niistä on saatu hyviä kokemuksia [30]. Terveyskioskien hyödyiksi on katsottu matala käyttökynnys, sekä niiden kustannustehokkuus kunnille, kun rutiininomaiset terveyskäynnit voitaisiin ohjata terveyskioskeihin ja vaikeat toimenpiteet terveysasemille [18].

Ajantasaista tilastoa terveyskioskien määrästä Suomessa ei ollut saatavilla tätä työtä kirjoitettaessa. Toisaalta, Yhdysvalloissa kioskeja on pelkästään Pursuant Health -nimisellä yrityksellä noin 3400 kappaletta. Muita terveyskioskeja tuottavia yrityksiä ovat muun muassa Wellpoint ja Stayhealthy.

Tämän työn tarkoituksena on kasata terveyskioski. Kioskin runko on tilattu Aalto-yliopiston Taiteiden ja suunnittelun korkeakoulusta. Kioskissa käytettävä ohjelmisto pohjautuu Juha-Matti Santeron diplomityön aiheeseeen. Koska työn aiheena olevan terveyskioskin on tarkoitus palvella käyttäjiä itsenäisesti julkisella paikalla, kioskin tulisi olla käytettävyydeltään mahdollisimman yksinkertainen. Näin ollen suoritettavat mittaukset voivat olla vain noninvasiivisia.

Tässä diplomityössä kerrotaan ensin teoriaa yleisimmistä noninvasiiveista mittauksista ja miksi juuri näillä mittauksilla saa hyvän kokonaiskuvan käyttäjän terveydentilasta. Yleisimmät mitattavat suureet ovat sydämen sähkökäyrä, veren happisaturaatio (SpO2), kehon impedanssi ja sitä kautta saatava sisäelinrasva, paino sekä verenpaineen mittaus. Teorian jälkeen kerrotaan miten mittaukset on toteutettu tässä työssä.

Tässä työssä on toteutettu käyttäjän painon, veren happisaturaation, pulssin ja verenpaineen mittaus. Sydämen sähkökäyrän mittaaminen ilman osaavaa henkilökuntaa on nykyisellään vielä liian monimutkaista luotettavien tulosten saamiseksi. Lisäksi kuivaelektrodien käyttö sähkökäyrän mittaamisessa on liian häiriöaltista johtuen ihon suuresta impedanssista. Näistä syistä itse EKG:n mittaus on päätetty jättää pois työstä, mutta sen ohjelmointi on toteutettu koodissa.

Lopuksi pohditaan miten tuolia voi jatkokehittää esimerkiksi parantamalla tuolin käyttöliittymää, lisäämällä rekisteröitymismahdollisuus ennen mittausta sekä tietojen mahdollista tallentamista esimerkiksi Taltioni-järjestelmään.

2 Fysiologiset ominaisuudet

2.1 Paino

Painon mittausta on pitkään käytetty yhtenä terveyteen liittyvistä parametreistä. Liikalihavuus on yhdistetty moniin sairauksiin ja ennenaikaisiin kuolemiin [4]. Paino ei kuitenkaan anna tarpeeksi hyvää kuvaa potilaan terveydentilasta, koska paino ei anna mitään tietoa esimerkiksi sisäelinten rasvamäärästä. Pelkkää painoa tarkempi arvo on painoindeksi (BMI). Sitä on käytetty myös liikalihavuuden mittaamiseen. Painoindeksi saadaan kaavalla: $BMI = w/h^2$, missä w = henkilön paino (kg) ja h = henkilön pituus (m). BMI lukemaa tulkitaan seuraavalla tavalla.

BMI	**Painoarvo**
Alle 18.5	Alipainoinen
18.5-24.9	Normaali tai tervepainoinen
25.0-29.9	Ylipainoinen
30.0 ja yli	Sairaaloisen ylipainoinen

BMI arvon ja painoarvon välinen yhteys aikuisille [4].

2.2 Verenpaine

Tavallinen sydän pumppaa yli 14000 litraa verta päivässä. Sydän sijaitsee pallean yläpuolella keuhkojen välissä. Sydämessä on kaksi suurta kammiota ja kaksi eteistä. Sydämeen saapuva veri tulee ensin eteiseen, josta se siirtyy kammioon ja sieltä edelleen takaisin verenkiertoon. Verenpaine muodostuu kahdesta osasta. Ensimmäinen ja korkeampi paine, niin sanottu systolinen, johtuu kammion pumppauksesta, jolloin veri työntyy aortaan ja sieltä edelleen eteenpäin kehossa. Alipaine, niin sanottu diastolinen, muodostuu systolisen paineen jälkeen, kun sydämen kammiot ovat levossa [11].

Verenpaine mitataan yleisimmin kädessä olevan hauislihaksen päälle sijoitetulla mansetilla. Yläpaineen mittaus suoritetaan siten, että mansetti täytetään ilmalla kunnes verisuonet tukeuttuvat ja veren kulku estyy. Tämän jälkeen mansetista lasketaan ilmaa kunnes sensori rekisteröi veren virtauksen suonissa. Alipaine saadaan tämän jälkeen siten, että ilmaa päästetään pois kunnes veren virtauksesta ei kuulu enää ääntä. Tämä transitiovaihe kertoo diastolisen eli alipaineen.

Verenpaineen nousu yli 115/75mm Hg on yhdistetty kasvavaan riskiin saada kardiovaskulaarinen sairaus [10]. Verenpaineen mittaamisessa hankaluuksia aiheuttaa yleensä mansetin oikeaoppinen kiinnittäminen. Mansetin pukeminen vaatii yleensä potilaan paidan riisumisen tai vähintään hihan ylös käärimisen. Lisäksi tiukasti tai löysästi kiinnitetty mansetti voi antaa vääriä tuloksia [13]. Markkinoille on kuitenkin tulossa erilaisia laitteita, joissa mansettia ei tarvita lainkaan [27]. Laitteen kiinnitys olisi paljon helpompaa käyttäen esimerkiksi ranneketta. Nämä laitteet pystyvät mittaamaan verenpainetta reaaliajassa [7], mihin mansetti ei pysty. Tällaiset ratkaisut voisivat mahdollistaa sen, että älykellotkin pystyisivät mittaamaan käyttäjän verenpaineen.

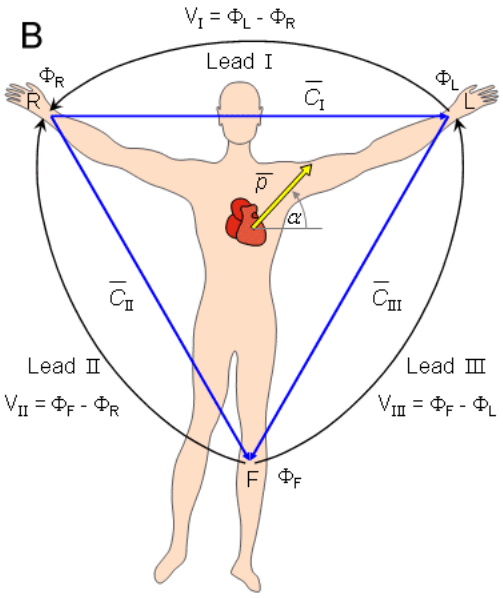
2.3 Sydämen sähkökäyrä

Willem Einthoven julkaisi ensimmäisen EKG-mittausjärjestelyn vuonna 1908. Einthovenin kehittämä kolmiojärjestely toimii nykyisen EKG-mittauksen pohjana [14]. Kuten kuvasta 2 nähdään Einthovenin kolmio mittaa kolmea eri johdinta.

$V_I = \Phi_L - \Phi_R$
$V_{II} = \Phi_F - \Phi_R$
$V_{III} = \Phi_F - \Phi_L$

Missä:

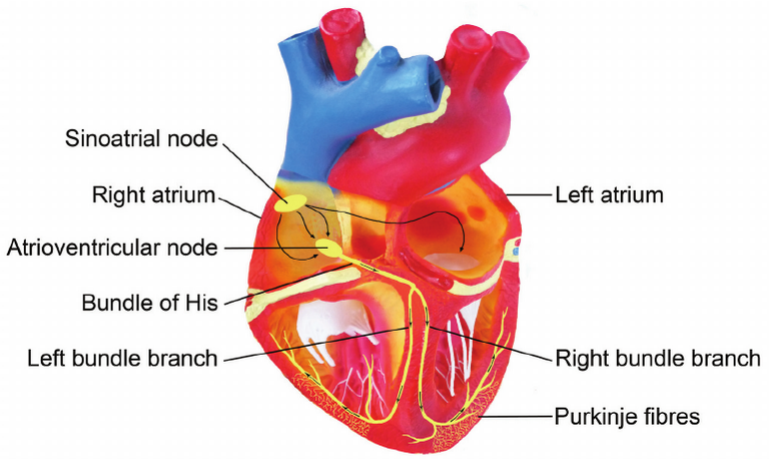
V_I on oikean ja vasemman käden potentiaaliero.
V_{II} on vasemman jalan ja oikean käden potentiaaaliero.
V_{III} on vasemman käden ja jalan potentiaaliero.
Φ_L on vasemman käden potentiaali.
Φ_R on oikean käden potentiaali.
Φ_F on vasemman jalan potentiaali.



Kuva 2: Einthovenin kolmioasettelu.

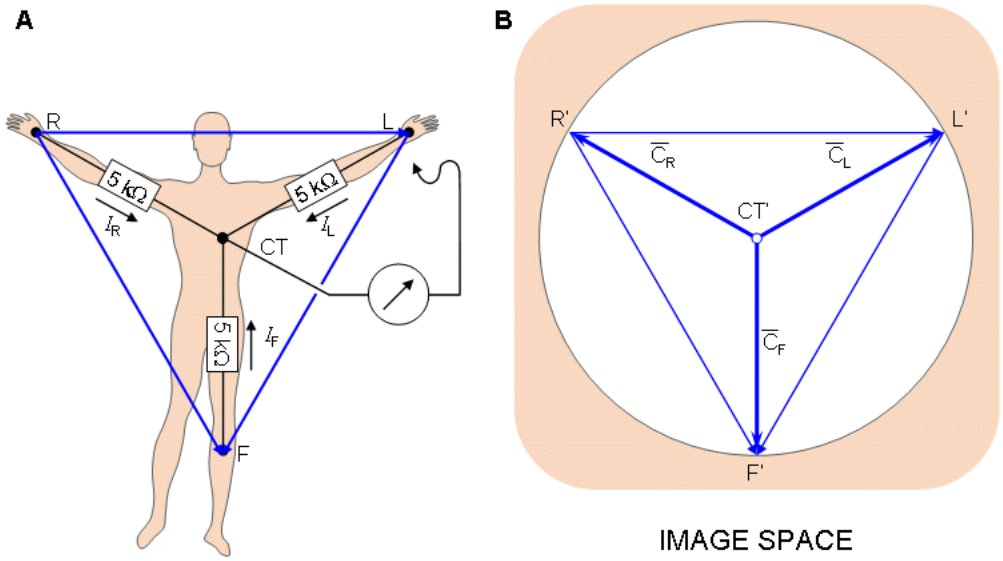
Kun sydän pumppaa verta sen eri osat aktivoituvat eri aikaan. Aktivaatioalue lähtee liikkeelle sydämessä sijaitsevasta sinussolmukkeesta. Sinussolmukkeesta aktiovaatioalue etenee sydämen eteisiin. Tämän jälkeen aktivaatioalue etenee eteiskammiosolmukkeen kautta sydämen kammioihin, kuva 3.

Eri aikoihin tapahtuvat aktivaatiot muodostavat potentiaalieroja, jotka projisoituvat Einthovenin johdinvektoreihin. Myöhemmin Frank Norman Wilson lisäsi Einthovenin järjestelmään keskiterminaalin, joka muodostettiin lisäämällä 5kΩ vastus jokaisen kolmen johtimen väliin, kuva 4. Näin saadaan mitattua kolme arvoa



Kuva 3: Sydämen eri aktivaatioalueet. [12]

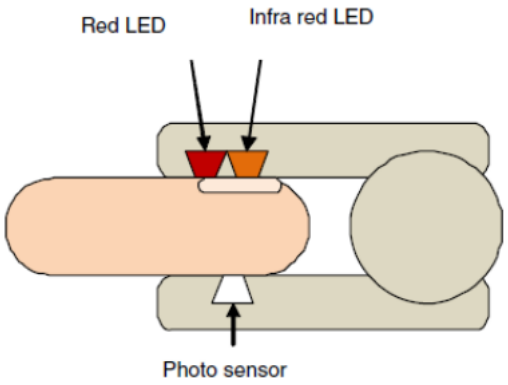
lisää: V_R, V_L ja V_F. Nämä johtimet mittaavat jännitettä käyttäen referenssinä wilsonin keskiterminaalia. Nykyisin käytetään yleisesti yhteensä 12 elektrodia, jotta potentiaaliero saadaan mitattua myös syvyyssuunnassa.



Kuva 4: Einthovenin kolmoisasettelu johon on lisätty Wilsonin keskusterminaali .

2.4 Veren happisaturaatio

Veri sisältää neljää eri tyyppistä hemoglobiinia [19], ja näistä kahta, oksyhemoglobiinia ja deoksyhemoglobiinia, käytetään veren happisaturaation mittaamisessa. Veren happipitoisuus mitataan käyttämällä kahta eripituista valon aallonpituutta: 660nm ja 940nm. Nämä kaksi valoa lähetetään emitteristä vastaanottimeen niin, että emitteri ja vastaanotin sijaitsevat saman kehon osan eri puolilla, esimerkiksi emitteri korvalehden edessä ja vastaanotin korvalehden takana, kuva 5.

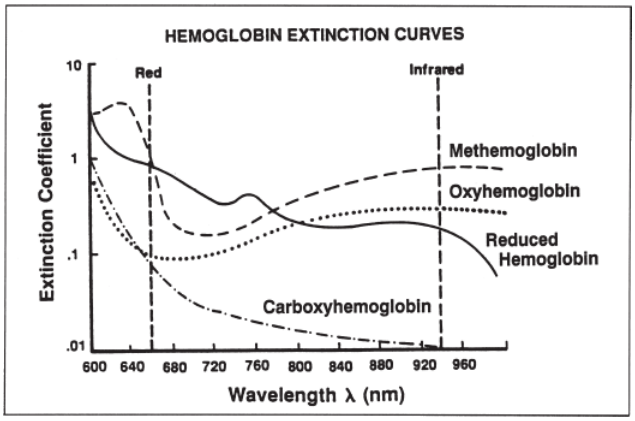


Kuva 5: Havainnekuva happisaturaation mittaamisessa käytettävästä anturista [23].

Happimolekyyli on sitoutunut oksyhemoglobiiniin, joten oksyhemoglobiini absorboi enemmän 940nm valon aallonpituuksia ja päästää 660nm aallonpituuksia läpi. Deoksyhemoglobiini, josta taas puuttuu happimolekyyli, absorboi enemmän 660nm aallonpituutta ja päästää läpi 940nm aallonpituutta. Kuten kuvasta 6 nähdään, näiden absorboitujen valon aallonpituuksien suhteista voidaan laskea veren happipitoisuus. Happipitoisuus saadaan kaavalla

$$SO_2 = \frac{cO_2Hb}{cO_2Hb + cHHb} \tag{1}$$

missä cO_2Hb on happea kuljettavan hemoglobiinin osuus (oksyhemoglobiini) ja $cHHb$ on redusoitu hemoglobiini [22].

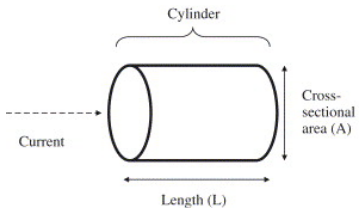


Kuva 6: Hemoglobiinin absortoivuuskerroin eri aallonpituuksille [19].

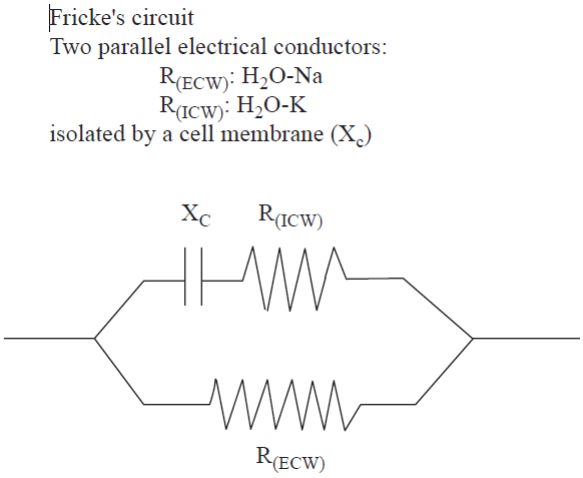
2.5 Kehon impedanssi

Kehon impedanssilla voidaan mitata kehon sisäisen veden jakautumista solun sisäiseen veteen (ICW) ja solun ulkopuoliseen veteen (ECW). Näitä arvoja mittaamalla voidaan arvioida kehon rasvattoman kudoksen osuus. Kehon voi mallintaa yksinkertaisimmillaan kuten kuvissa 7 ja 8. Näitä ominaisuuksia käyttämällä, yhdessä

henkilön iän, sukupuolen ja etnisen alkuperän kanssa, voidaan arvioida kehon koostumus [5].



Kuva 7: Kehon volyymin impedanssin mallinnus yksinkertaisen sylinterin avulla [5]



Kuva 8: Yksinkertainen piirikaaviomallinnus solun rakenteesta. Kuvassa X_c kuvaa soluseinämän kapasitanssia, R_{ICW} solun sisäistä resistanssia ja R_{ECW} solun ulkoista resistanssia [5]

Impedanssin mittaamisessa voidaan käyttää yksitaajuusanalyysiä (SF-BIA) tai monitaajuusanalyysiä (MF-BIA). Impedanssi mitataan yleensä käden ja jalan välillä, mutta jotkut instrumentit voivat käyttää myös jalasta jalkaan tai kädestä käteen mittausta [5]. Yksilöille, joiden nestetasapaino on normaali, BIA antaa hyvän arvion rasvattoman kudoksen (FFM) ja kehon kokonaisvesimäärän (TBW) suhteelle. Mittaamisessa on varmistettava, että käytetään oikeaa ikään ja populaatioon liittyviä yhtälöitä. Mikäli potilaan nestetasapaino on järkkynyt BIA-analyysin on todettu antavan epäluotettavia tuloksia [20].

Yksitaajuusanalyysissä käytetään yleensä 50kHz, jolloin sillä mitataan solujen ulkopuolisen veden (ECW) ja solujen sisäisen veden painotettua summaa. Monitaajuusanalyysissä käytetään yleensä taajuuksia 0, 1, 5, 50, 100, 200 ja 500kHz evaluoidessa FFM, TBW, ICW ja ECW osuuksia. Lisäksi mikäli resistanssin ja kapasitanssin yhteydestä muodostaa graafin voidaan bioimpedanssi vektorianalyysillä selvittää mahdollisia epänormaalisuuksia kehossa [15].

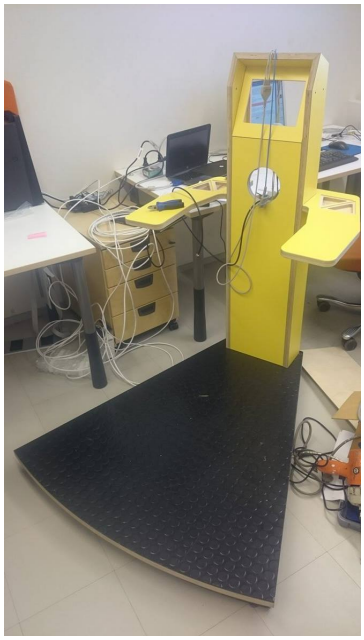
3 Terveyskioskin suunnittelu

Tässä työssä toteutettiin terveystuolissa istuvan käyttäjän painon, veren happisaturaation ja verenpaineen mittaus. Käyttäjän painon mittaus toteutettiin käyttämällä kolmea QL-11A painesensoria [21]. Nämä painesensorit on liitetty Arduinon HX711 ad -muuntimeen [1], johon on saatavilla internetistä valmiit kirjastot https://github.com/bogde/HX711. HX711-muunnin on puolestaan liitetty Arduinon proto levyyn https://www.arduino.cc/en/Main/ArduinoProtoShield ja sitä kautta Arduino Yuniin. Arduino Yun valittiin sen sisäänrakennetun wifin takia, mutta mikä tahansa muu yhteensopiva piirilevy käy.

Muut fysiologiset mittaustiedot saadaan käyttämällä BerryMedin Patient Monitor PM6750 -moduulia [25]. Tämä moduuli kommunikoi tietokoneen kanssa käyttäen hexadesimaali bittijonoja. Kommunikaation hoitaa python skripti, joka on alunperin kehitetty Juha-Matti Santeron diplomityössä [9].

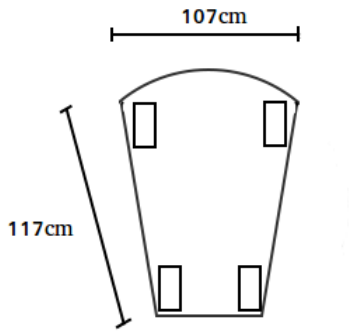
3.1 Kioskin tausta

Tässä työssä käytettävä terveyskioski on rakennettu Aalto-yliopiston Taiteiden ja Suunnittelun korkeakoulussa projektityönä, kuva 9. Kuvissa 10 ja 11 on nähtävissä kioskin mitat. Kioskin on tarkoitus jatkaa Aalto Health Factoryn Terttu terveystuolien kehityskaarta aina mahdollisesti kaupalliseksi sovellukseksi asti. Tämän kioskin tarkoitus olisikin palvella asiakkaita esimerkiksi Aalto-yliopiston aulassa tai jossakin muussa julkisessa tilassa.

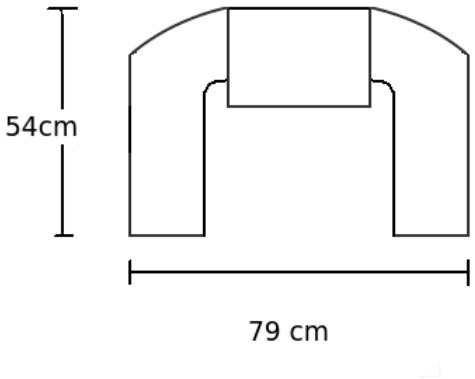


Kuva 9: Terveyskioskin runko.

Merkittävä kehitysaskel verrattuna aikaisempiin Terttu-terveystuoleihin on siinä, että aikaisemmat Terttu-tuolit ovat tallentaneet mitattavien henkilöiden tiedot



Kuva 10: Terveystuolin pohja, paineantureiden paikka on kuvattu neliöillä.



Kuva 11: Terveystuolin torni.

tuolissa olevaan tietokoneeseen. Tässä työssä käytettävällä internet-pohjaisella käyttöliittymällä voidaan mitattavien henkilöiden tiedot tallentaa erilliselle palvelimelle tai johonkin muuhun järjestelmään, kuten esimerkiksi Taltioni-järjestelmään. Näin käyttäjästä mitattuja tietoja pääsisivät tutkimaan esimerkiksi terveydenhuollon ammattilaiset tai henkilö itse.

3.2 Tulevaisuuden tavoitteet

Tässä kappaleessa pohditaan enemmän kioskin mekaanisia kehityskohteita. Käyttöliittymään liittyviä kehitysideoita käsitellään myöhemmin. Tämä versio terveskioskista on aivan liian painava. Seuraavaat versiot täytyisi suunnitella kevyemmistä materiaaleista. Lisäksi konfiguraatiota on hankala liikutella, se ei muun muassa mahdu ovista kuin ottamalla pohjaosan irti. Terveyskioskikonfiguraation pohjalevyyn olisi hyvä saada esimerkiksi taitettavat pyörät, jolloin sen kuljettaminen olisi vaivattomampaa. Lisäksi painonmittauksen kannalta olisi tiedettävä paljonko pohjalevy painaa itsessään. Näin voitaisiin parantaa mittaustarkkuutta.

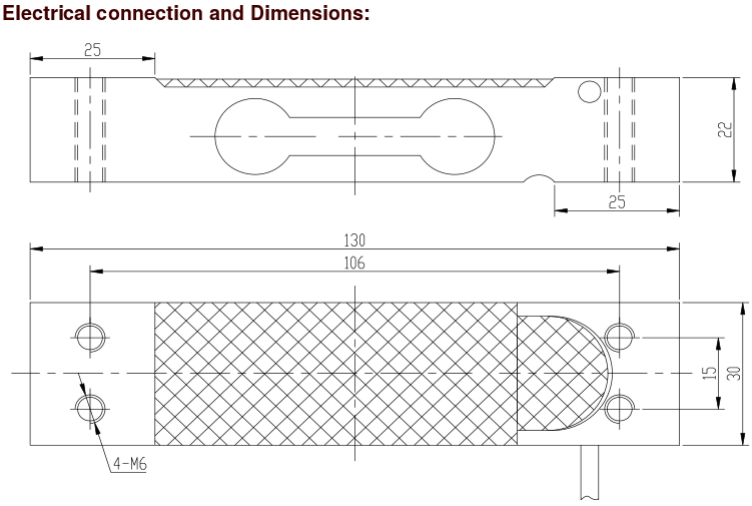
Seuraavaan versioon olisi myös hyvä saada liikuteltavat käsinojat. EKG-elektrodien paikka pitää myös selvittää. Nykyisessä konfiguraatiossa antureiden paikka on liian

Kuva 12: EKG-elektrodeille suunnitellut paikat tuolin käsivarsissa.

edessä. Ne ovat käyttäjän kämmenien alla, vaikka niiden pitäisi olla käsivarsien alla, kuva 12. Tuoliin olisi myös hyvä lisätä jonkinlaiset kahvat, joista kohdehenkilö voisi ottaa rennosti kiinni käsillä. Näin estettäisiin tahaton käsivarsien heiluminen, mikä puolestaan hankaloittaa EKG-mittauksia.

4 Mittausjärjestely

Mittausten suorittamisessa käytettävä BerryMed PM6750 -potilasmonitori [25] mittaa automaattisesti veren happisaturaatiota ja sydämen pulssia. Käyttäjä voi pyytää mittaamaan verenpaineen erikseen. Painon mittaus tapahtuu neljällä QL11A -painemoduulilla [21], jotka ovat yhdistetty Arduinon HX711 24-bittiseen AD-muuntimeen [1], kuvan 10 mukaisesti. Sensorit on aseteltu pohjalevyn alle reunoihin. Näin yksittäisen sensorin kuormitus ei kasva keskimäärin liian suureksi.



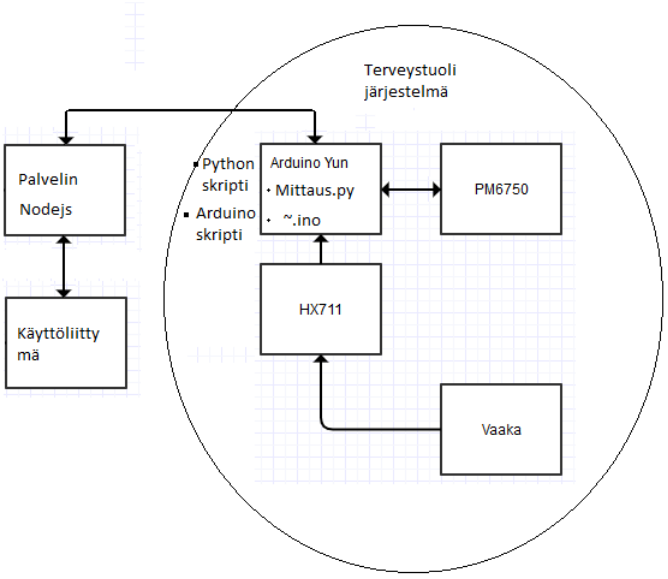
Kuva 13: QL-11A painesensorin rakenne [21].

Tässä työssä käytettävä ohjelmisto on alunperin kehitetty Santeron [9] diplomityössä. Ohjelmistoa on jatkokehitetty siten, että nykyisellään siihen kuuluu neljä osaa, kuva 14.

- Arduinon ATmegassa ajettava .ino skripti.
- Arduino Yunissa ajettava python skripti.
- Palvelimella oleva javascript serveri.
- Html käyttöliittymä.

Järjestelmä toimii siten, että Arduino Yunissa pyörii python skripti *mittaus.py*. Tämä skripti kommunikoi sekä PM6750-moduulin kanssa että HX711 ADC -moduulin kanssa. PM6750-moduulin kanssa kommunikointi tapahtuu hexadesimaalimerkkijonoina, joista *mittaus.py* rakentaa json-tietueen lähetettäväksi palvelimelle. Painon miHX711-moduulin kanssa kommunikointi tapahtuu siten, että Arduino Yunin ATmega prosessorilla ajetaan arduino skriptiä. Arduino skripti käyttää ilmaiseksi internetistä saatavaa *HX711.h* kirjastoa [2] analogisen painosignaalin muuntamiseen digitaaliseksi. Kommunikointi arduino skriptin ja python skriptin välillä tapahtuu käyttäen Bridge-kirjastoa.

Palvelimella pyörivä javascript on ohjelmoitu niin, että siihen voi yhdistää useampia terveystuoli-järjestelmä. Tällaista järjestelmää ei ole vielä testattu, joten lisätyötä on tehtävä mikäli sellainen halutaan toteuttaa tulevaisuudessa. Lisäksi tallennusmahdollisuutta ei toteuteta tässä työssä. Tietojen erilaisia tallennusmahdollisuuksia pohditaan tämän työn lopussa.



Kuva 14: Mittausjärjestelmä pohjautuen [9] diplomityöhön.

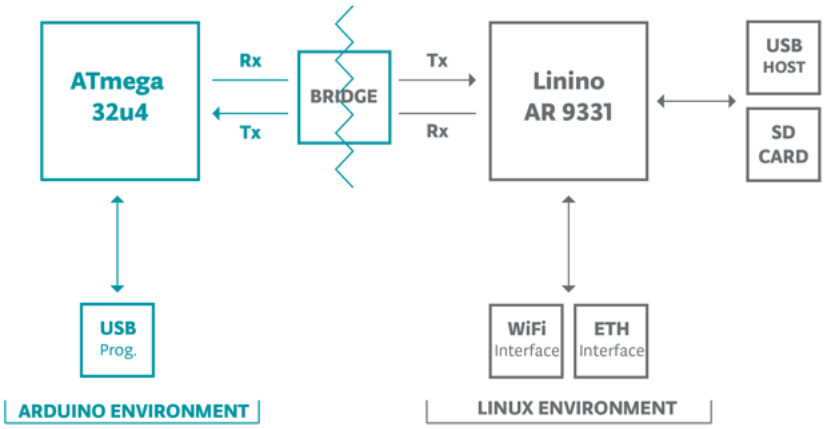
4.1 Arduino Yun

Arduino Yuniin https://www.arduino.cc/en/Main/ArduinoBoardYun pitää asentaa kmod-usb-serial ja kmod-usb-serial-pl2303 paketit käyttäen opkg pakettien hallintajärjestelmää. Ilman niitä PM6750 moduulia ei voi lukea usb-serial muuntajaa käyttäen. Lisäksi, jotta Arduinon sketchi toimisi parhaiten se on ajettava aina, kun Arduinon käynnistää uudestaan.

4.2 PM6750 moduuli

PM6750-moduulia käytetään veren happisaturaation, pulssin, sydämen sähkökäyrän sekä verenpaineen mittaamiseen. Moduuli on mahdollista liittää tietokoneeseen joko bluetoothilla, USB Standardi-B:tä käyttämällä tai serial-liitännällä. Tässä työssä moduuli liitettiin käyttäen usb-serial adapteria sillä USB-B liitäntä vaatisi PM6750-moduulin omat ajurit, joita ei ole saatavilla linuxille. Moduulin käyttö on verraten yksinkertaista. Moduuli lähettää ja vastaanottaa paketteja heksadesimaaleina, näistä paketeista python skripti parsii komentoja edelleen serverille. Jokainen paketti koostuu viidestä osasta.

1. Kahden tavun pituinen tunniste.



Kuva 15: Arduino Yunin kaaviokuva.

2. Paketin pituus.
3. Paketin tyyppi, esimerkiksi EKG tai Verenpaine.
4. Itse paketin sisältö.
5. Tarkistussumma.

Taulukossa 1 on nähtävissä esimerkkipaketti moduulilta tietokoneeseen. Paketti kuvaa EKG käyrää eri kanavilta. Paketin kaksi ensimmäistä osaa ovat tunnisteet $0x55$ ja $0xAA$. Tunnisteiden jälkeen on paketin pituus $N = n+2$, missä n on paketin dataosan pituus tavuissa. Sen jälkeen on paketin tyyppi, jonka koodi $0x01$ viittaa tässä tapauksessa EKG aaltoon. Seuraavana on itse sisältö ja viimeisenä on tarkistussumma, joka lasketaan kaavalla $SUMMA = NOT(N + A1 + ... + An)$.

Tunniste		Pituus	Tyyppi	Data $A1..n$	Tarkistussumma
0x55	0xAA	0x0A	0x01	0x80 ... 0x80	0x74

Taulukko 1: Esimerkkipaketti

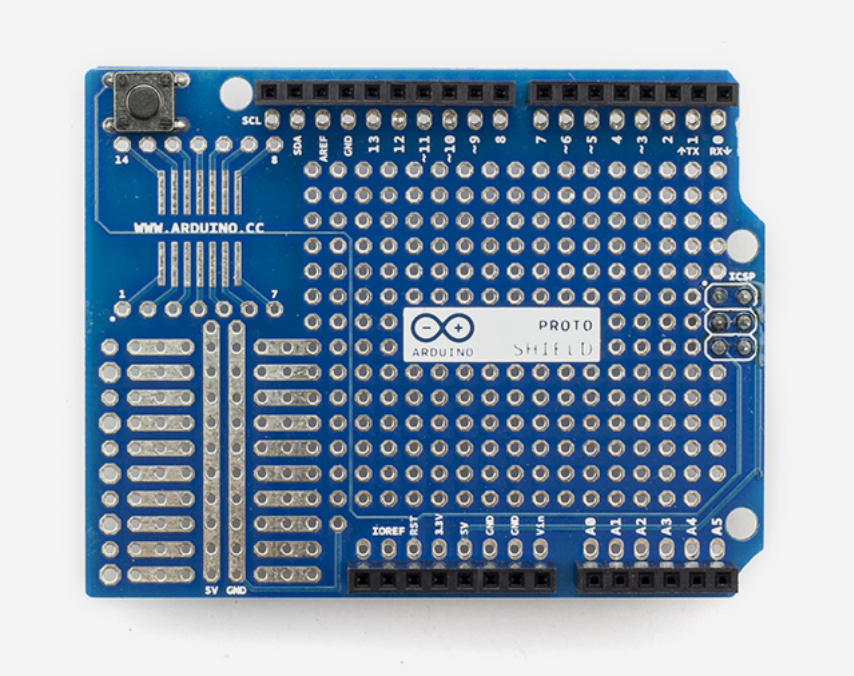
Python skripti käsittelee PM6750-moduulilta tulevaa jatkuvaa pakettivirtaa siten, että skripti ottaa bufferiin 200 tavun pituisen jonon, jonka se käy läpi ja siirtyy sen jälkeen taas odottamaan, että bufferi täyttyy. Tarkistussumman hyödyntämistä ei käytetä tässä työssä, koska sen toteuttaminen ei nähty olevan oleellinen osa työtä. Tämä siitä syystä, että terveyskioskista saatava data ei ole verrattavissa sairaalaympäristössä tapahtuviin mittauksiin, joissa potilaan henki saattaa pahimmassa tapauksessa vaarantua mikäli virheellinen paketti sotkisi jonkin mittauksen.

4.3 Painon mittaus

HX711-moduulissa on kaksi sisääntulokanavaa A ja B. Kanava A päätettiin valita, sillä siinä on suurempi vahvistus kuin kanavassa B. Neljä painesensoria on kiinnitetty terveystuolin pohjaan ja niiden johdot on kolvattu HX711-moduulin sisääntuloon

siten että punainen on kolvattu plus sisääntuloon, musta miinus sisääntuloon, ja vihreä ja valkoinen signaaleihin. HX711-moduuli osaa automaattisesti laskea painesensoreilta tulevan datan ja keskiarvoistaa sen, jolloin riittää että Arduinon skriptissä kutsutaan HX711.h kirjaston get_unit -funktiota. Tämä funktio palauttaa sensoreilta tulevan arvon.

Arduinon skriptiin on lisätty kerroin, joka muuttaa painesensorilta tulevat arvot kiloiksi. Kerroin näyttää henkilön painon kilon tarkkuudella. Tämä on tarpeeksi riittävä tähän työhön. Tarkempaa kerrointa varten täytyisi tietää muun muassa terveystuolin alustan paino, joka ei ole tiedossa. Lisäksi, kioskin käyttäjillä tulee olemaan vaatteet päällä mittaushetkellä, joten kilon tarkkuus nähtiin riittäväksi. Tarkkuuden muuttaminen myöhemmin tarpeen vaatiessa on yksinkertaista, mikäli siihen on tarvetta.



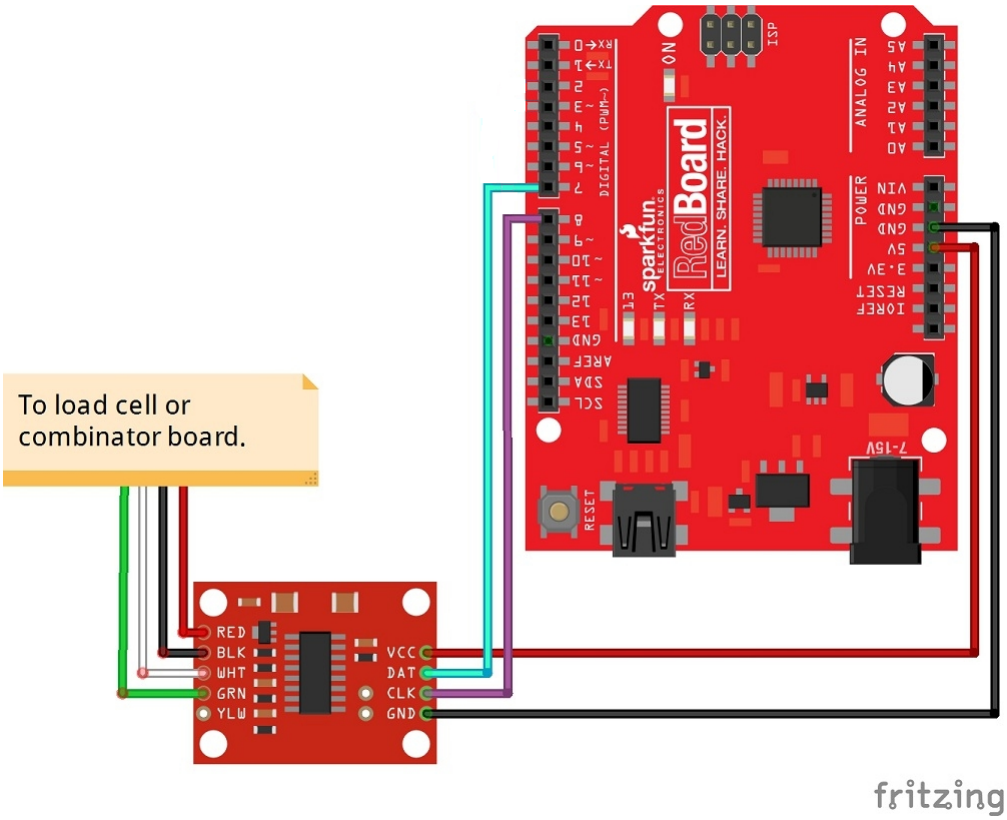
Kuva 16: Arduino Protoshield levy.

Painoindeksin laskenta suoritetaan siten, että käyttäjä syöttää oman pituutensa senttimetreinä sille varattuun tilaan, josta se sitten lasketaan käyttäen vaa'alta saatua painoa apuna.

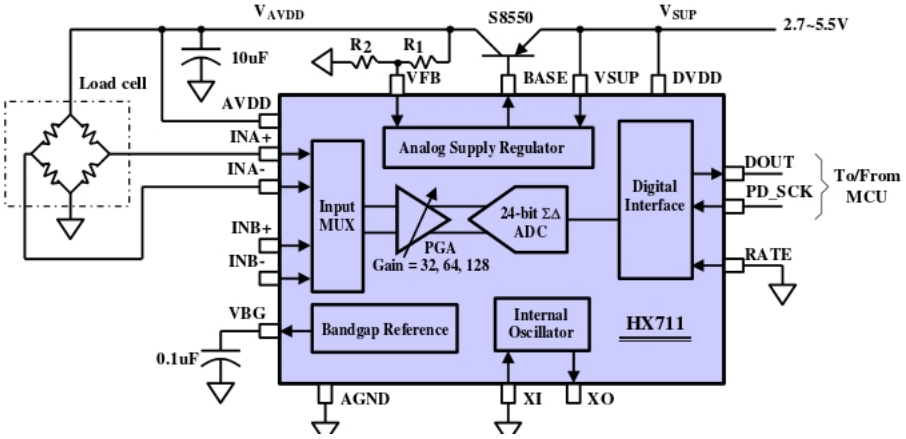
4.4 EKG

Sydämen sähkökäyrän mittaaminen on normaalioloissa hankalaa. Tässä työssä oli ensin ajatus käyttää kuivaelektrodeja. Ne muodostavat yhdessä ihon kanssa liian suuren lähtöimpedanssin, mikä tuottaisi aivan liikaa kohinaa vahvistimeen [6].

Kuvasta 19 nähdään, $V_{in} = \frac{Z_{in}}{Z_{in}+Z_{out}}V$. Jotta signaali siirtyisi vahvistimelle mahdollisimman häiriöittä tulisi olla $V_{in} \approx V$. Tämä saavutetaan vain kun $Z_{in} \gg Z_{out}$.

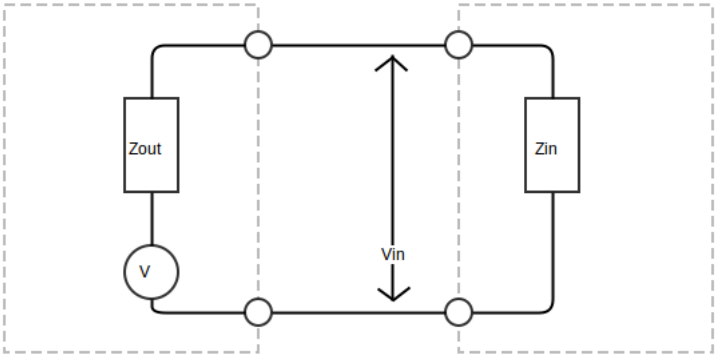


Kuva 17: Paineantureiden juottaminen HX711 AD -muunnintimeen ja sen liittäminen Arduino Protoshield -levyyn.https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide. Kuvaa muokattu.



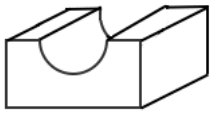
Kuva 18: HX711 AD -muunnin.

Teoriassa kasvattamalla Z_{in} äärettömän suureksi ei tarvitsisi huolehtia Z_{out}:in arvosta, mutta tällöin kaikki signaaliin tulevat virheet moninkertaistuisivat ja näkyisivät ulostulossa, mikä ei ole toivottavaa. Näin ollen ihon ja elektrodin välille olisi pystyttävä muodostamaan mahdollisimman vähäimpedanssinen liitäntä. Elektrodin olisi myös pysyttävä henkilön ranteissa tukevasti kiinni, koska käsien liikuttelu voisi



Kuva 19: Lähtöimpedanssin ja tuloimpendanssin yhteys. Tässä tapauksessa $V ja Z_{out}$ kuvaavat ihon elektrodia ja Z_{in} vahvistimen tuloliitäntää [6].

tuottaa artefakteja signaaliin. Tähän tarkoitukseen sopisi esimerkiksi hydrogeelielektrodi, joka olisi kiinnitetty kuvan 20 rakoon.

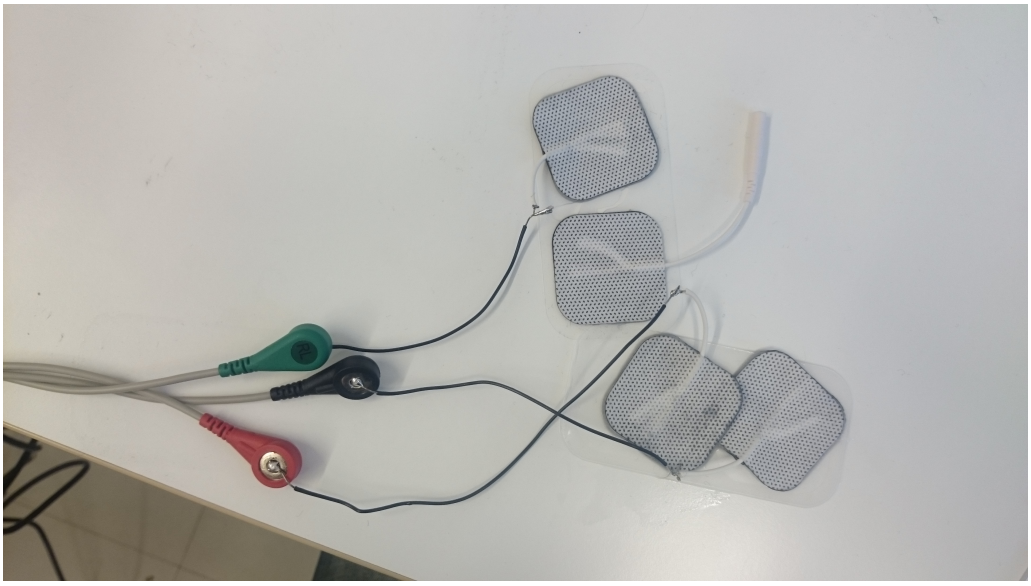


Kuva 20: Mahdollinen EKG elektrodien pidike. Mitattavan henkilön ranne tulisi rakoon, johon elektrodit olisi kiinnitetty.

Hydrogeelielektrodien käyttö olisi suositeltavaa, sillä niiden kiinnittäminen ja irroittaminen on vaivatonta ja nopeaa. Lisäksi hydrogeelin kosteus lisää elektrodin ja ihon yhteistä pinta-alaa ja vähentää ihon uloimman kerroksen eli marraskesin (stratum corneum) suurta impedanssia [16]. Hydrogeelien huonona puolena on tosin niiden melko nopea likaantuminen ja niiden märkä pinta voi tuntua joistakin epämiellyttävältä. Perinteisiä EKG-elektrodeja käyttämällä saataisiin tarkemmat mittaustulokset, mutta niiden kiinnittäminen on hankalaa ilman osaavaa henkilökuntaa. Lisäksi ne on vaihdettava jokaisen mittauksen jälkeen.

Kokeilimme ensimmäiseksi Clas Ohlsonilta saatavia tens-elektrodeja http://www.clasohlson.com/uk/Tens-Electrodes/34-4103-1. Näiden elektrodien impedanssi oli kuitenkin megaohmien luokkaa, mikä on aivan liikaa vaadituilta elektrodeilta. Lisäksi tens elektrodien käyttö olisi käyttäjälle epämieluisa kokemus niiden märkyyden takia ja olisi otettava myös huomioon hygienia käyttäjien välissä. Myös elektrodien uudelleen kosteuttaminen on erittäin epäkäytännöllistä 21. PM6750 moduulin EKG-elektrodien tuloimpedanssit ovat luokkaa 10M ohmia. Ideaalisten elektrodien impedanssi saisi olla luokkaa 0.1M ohmia.

Seuraavassa taulukossa 2 on esitetty yhteenveto erityyppisistä EKG-elektrodeista.



Kuva 21: Tens elektrodit on kolvattu PM6750 moduulin EKG liittimiin.

EKG-elektrodi	+	-
Hydrogeeli	Pieni impedanssi Nopea kiinnittää	Epämiellyttävä iholle Voi irrota
Perinteinen	Pieni impedanssi	Hankala kiinnittää
Kuvaelektrodi	Mukava iholle Ei tarvetta kiinnittää	Suuri impedanssi Häiriöille altis

Taulukko 2: EKG-elektrodien vertailu

5 Ohjelmointi

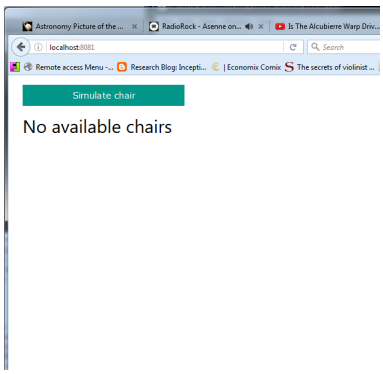
Tämä työ pohjautuu pääosin Juha-Matti Santeron diplomityöhön [9]. OpenWrt:ssä ajettavaan python scriptiin on lisätty HX711-sensorilta tuleva data, jota lähetetään Bridge-moduulin yli linuxin puolelle. Lisäksi koodiin on lisätty EKG-signaalin jokainen kanava, johtimen valinta, vahvistuksen säätö sekä taajuusalueen valinta. Arduinon .ino tiedosto käynnistää myös python skriptin siten, että python skriptille annetaan palvelimen osoite argementtina.

```
linuxScript.runShellCommandAsynchronously("python
/root/pyscripts/mittaus_v5.py -s <iposoite>");
```

Tällöin ei tarvitse ottaa selville Arduinon ip-osoitetta vaan riittää tietää mihin palvelimeen halutaan yhdistää.

5.1 Käyttöjärjestelmä

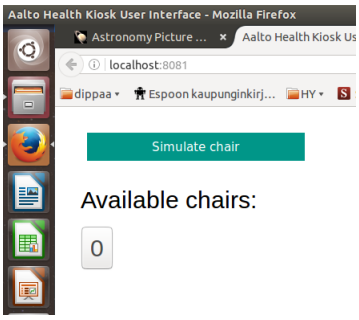
Käyttöjärjestelmä on selainpohjainen. Sen ulkoasun suunnittelussa on käytettty apuna w3.css-kirjastoa. Sivun toimivuutta mobiililaiteilla on kokeiltu vain Android-käyttöjärjestelmän laitteella. Sivujen saaminen mobiililaitteille toimivaksi vaatii vielä lisää työtä. Käyttöliittymä on nähtävissä kuvissa 23, 24 ja 25. Käyttöliittymän kieli on tällä hetkellä vielä kokonaan englanti, koska silloin myös suomea osaamattomat voivat käyttää kioskia alkuvaiheessa. Myöhemmin sivuille voi toki lisätä eri kielivaihtoehtoja.



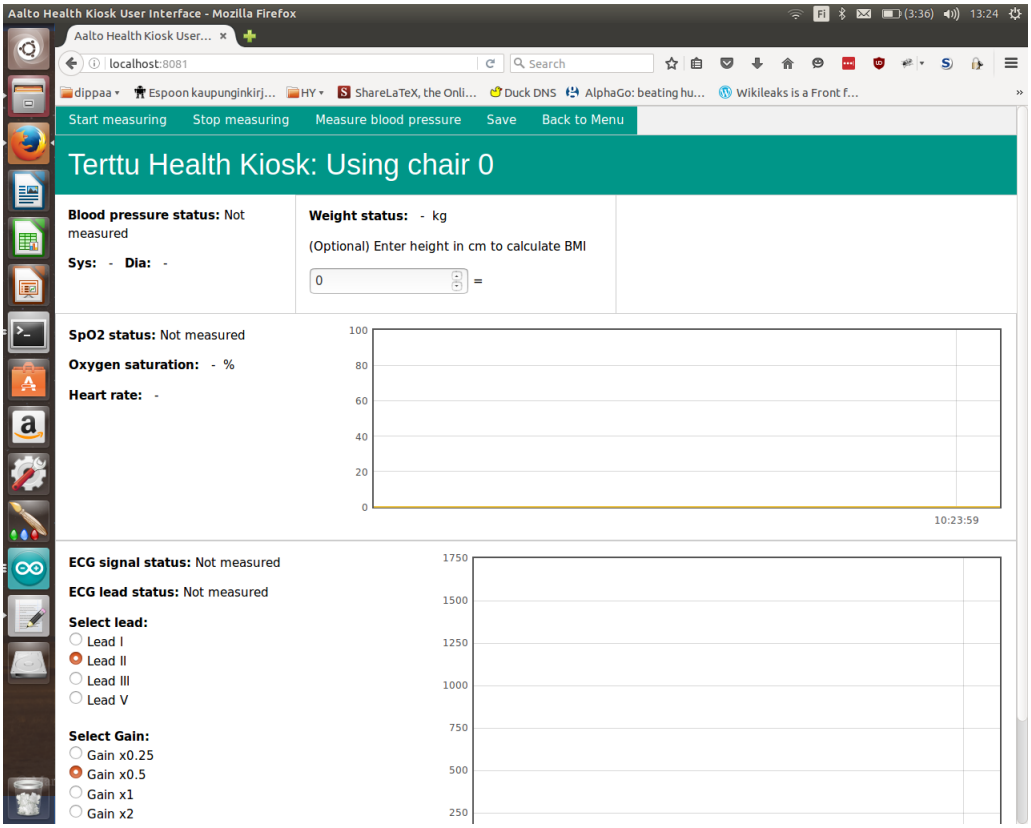
Kuva 22: Terveyskioskin käyttöliittymän valikko.

Sivun ulkoasua voi jatkosuunnitella entistä käyttäjäystävällisemmäksi esimerkiksi käyttämällä internetistä saatavia ilmaisia css tyylejä, esimerkiksi http://www.dynamicdrive.com/style/.

Nykyisin pilvipalvelujen käyttö on lisääntynyt muun muassa Dropboxin ja Google Driven myötä. terveyskioski tulisi tehokkaiten käyttöön, mikäli käyttäjillä olisi omat profiilit ja heidän tietonsa tallentuisivat jollekin pilvipalvelimelle esimerkiksi Taltioni-järjestelmään. Taltioni-terveystiliä ylläpitää Hyvis-ICT Oy [28]. Käyttäjän luomaan taltioni tiliin voidaan liittää myös erilaisia palveluja esimerkiksi Terveystalon Oma



Kuva 23: Terveyskioskin käyttöliittymän valikko, kun tuoleja on vapaana.

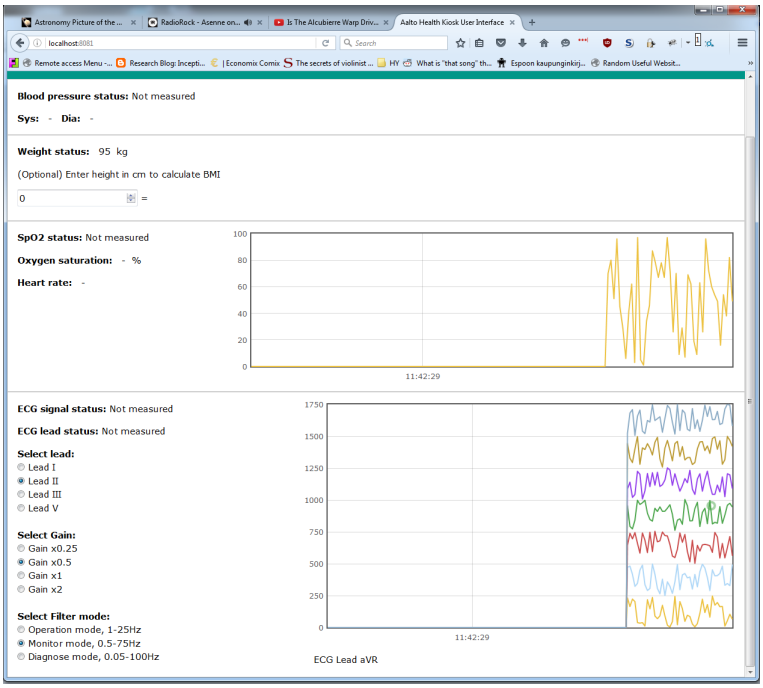


Kuva 24: Terveyskioskin käyttöliittymä.

Terveys -palvelu. Mikäli tuolin mittaamat tiedot tallennettaisiin Taltioniin, tarvitaan paljon jatkokehittelemistä ohjelmiston saamiseksi yhteensopivaksi järjestelmän kanssa.

5.2 Palvelimen valinta

Tätä työtä varten saatii Aalto IT:stä lainaan virtuaalinen Windows Server 2012, jossa palvelinohjelmiston oli tarkoitus pyöriä. Pian osoittautui kuitenkin, että Windows serverin saattaminen toimintakuntoon on erittäin haastavaa verrattuna Linux palvelimeen, jossa nodejs palvelimen saa ajokuntoon yhdellä komennolla ja tarvittavat



Kuva 25: Terveyskioskin käyttöliittymä simuloiduilla mittauksilla. EKG-liitännän kanavan nimi näkyy viemällä hiiri kyseisen kuvaajan päälle.

portit saa samoin helposti avattua esimerkiksi käyttämällä Uncomplicated Firewallia.

Tässä työssä tarvitaan tcp-porttia 6000 ja 8081. Tcp-portin 6000 kautta hoidetaan liikenne Arduinon ja palvelimen välillä ja 8081 hoitaa käyttäjien ja palvelimen välisen liikenteen. Tulevaisuudessa, jos palvelimen ja käyttäjien välinen liikenne halutaan salata siihen tarvitaan tcp-porttia 443.

5.3 Kehitysideoita

Nykyinen Arduinon .ino tiedosto täytyy ajaa joka kerta, kun Arduinoon käynnistää, jotta varmistutaan, että ohjelma toimii sujuvasti. Arduinon .ino tiedoston käynnistäminen käynnistää samalla python skriptin Linuxin puolelta. Seuraava kehitysaskel voisi olla koko python skriptin korvaaminen Arduinon idestä käsin ajettavalla .ino tiedostolla. Tämä skripti hoitaisi silloin kaiken kommunikaation niin serverille kuin PM6750 moduulille. Tämä voisi olla onnistua käyttämällä Arduinon Bridge-moduulia.

Terveystuoli tarvitsee lisäksi käyttöjännitteen, joka tällä hetkellä hoituu verkkovirralla. PM6750 moduuli tarvitsee DC 12V, 3A ja Arduino 5V jännitettä, jonka se saa micro-usb liittimen kautta. Koska terveystuolikokonaisuus on varsin painava ja hankalasti liikutettava, ei ole tarvetta hankkia erillisiä akkuja vaan käyttöjännitteen voi hoitaa verkkovirralla tuolin läheltä.

Arduinon yhteyttä palvelimelle ei myöskään ole salattu, eikä myöskään html liikennettä palvelimelle. Tässä on kehitettävä mahdollisten väärinkäytösten varalle

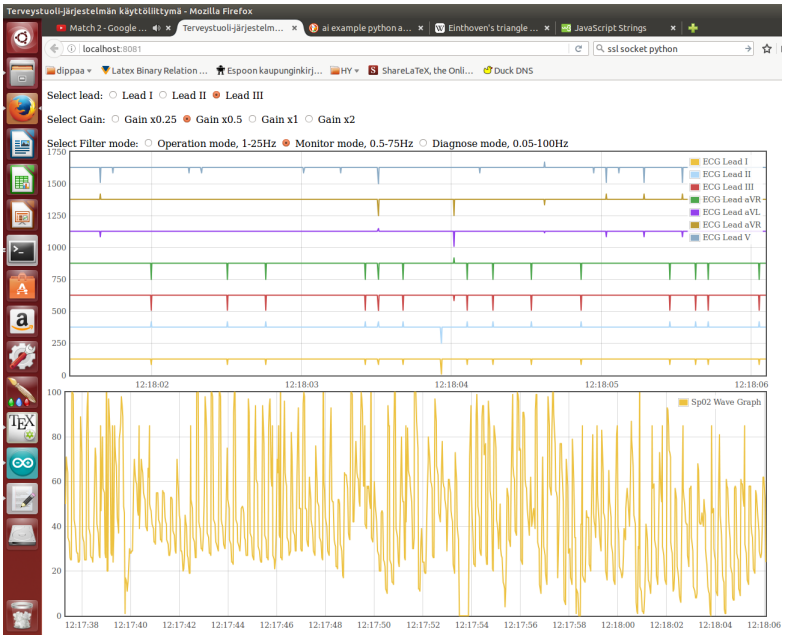
salattu liikenne tuolin ja palvelimen välillä kuin myös palvelimen ja webbisivun välille.

EKG:n mittaaminen tällaisella mittausjärjestelmällä on hankalaa, johtuen niin käyttäjien osaamisesta kiinnittää elektrodit, kuin myös elektrodin ja ihon välisestä impedanssista. Varsinkin jos halutaan luotettavia tuloksia myöhempää käyttöä varten. Tästä syystä EKG:n mittaaminen on tarpeetonta nykyisessä sovelluksessa. Ainakin niin kauan kunnes elektrodin ja ihon impedanssi saadaan tarpeeksi pieneksi ja käsille saadaan rakennettua jonkinlainen kehikko, missä niiden liike olisi minimaalista. Ideaaliset elektrodit olisivat vähäimpedanssisia kuivaelektrodeja, mutta sellaisia ei tämän työn tekohetkellä ollut saatavilla.

Toimivia mobiilisivuja varten on saatavilla internetistä erilaisia ilmaisia ja maksullisia ohjelmia joilla on mahdollista kääntää internetsivut mobiililaitteille sopiviksi. Näitä ovat esimerkiksi http://bmobilized.com/ ja https://www.dudamobile.com/. Näitä kokeiltaessa sivut eivät kuitenkaan toimineet halutusti, lisäksi nousee kysymys tietoturvasta. Parempi idea onkin saattaa sivut toimiviksi ilman kolmansia osapuolia.

6 Testaus

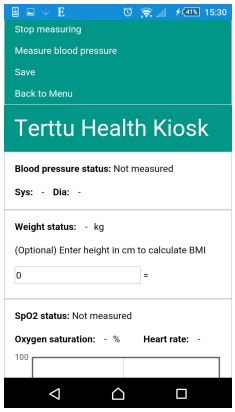
Ensimmäiset testien perusteella ohjelma toimii kaatumatta, mutta pitkän ajan päästä signaaleihin näyttäisi syntyvän häiriötä. Tämän häiriön nollautuminen kestää hetken ennen kuin mitattava signaali saadaan näkyviin. Senkin jälkeen saatava signaali näyttää aika häiriöiseltä, kuten kuvasta 26 nähdään.



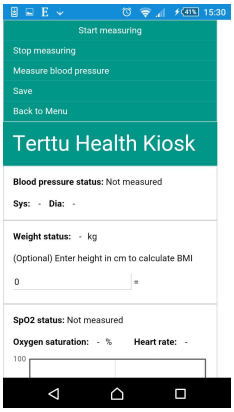
Kuva 26: Sp02 signaalin muoto, kun ohjelman on annettu olla päällä ennen mittausta. EKG signaalissa on myös nähtävissä häiriöitä.

Käyttöliittymän testaus Internet Explorer -selaimella toimii osittain. Tuolien simulaatiota kokeiltaessa huomattiin, että happisaturaatio- ja EKG-kuvaajat päivittyvät, mutta muuten data ei päivity serveriltä käyttöliittymään. Tämä on mielenkiintoinen ilmiö, sillä muilla selaimilla (Mozilla Firefox ja Chrome) kokeiltaessa kaikki kentät päivittyvät niin kuin pitääkin. Lisäksi, Internet Explorer ei anna konsoliin mitään virheilmoitusta, joten ongelma voi liittyä ie:n ominaisuuksiin.

Mobiiliversio käyttöliittymästä on myös vielä keskeneräinen ja siinä esiintyy muutamia bugeja. Esimerkiksi mobiililaitteen kääntäminen vääristää hieman /div elementtien kokoa näytöllä vrt. kuvat 27a ja 27b. Lisäksi kuvaajien kokoa pitäisi suunnitella käyttäjälle mieluisemmaksi.

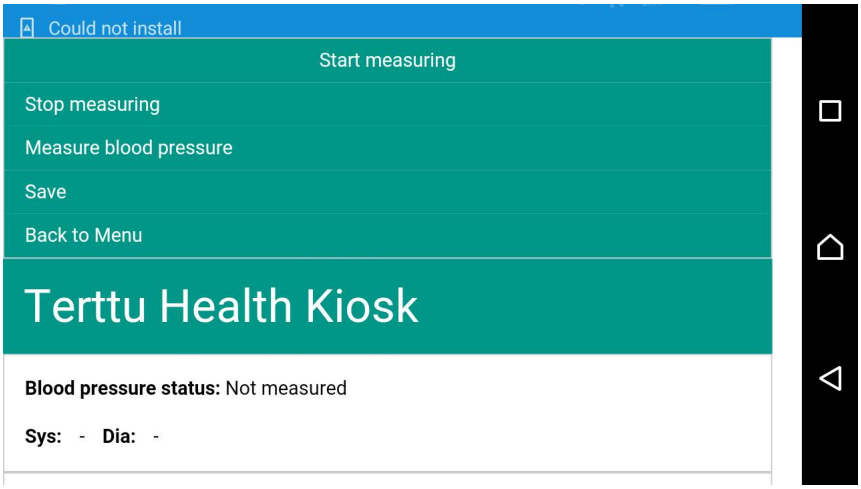


(a) Mittausvalikko ennen mobiililaitteen kääntämistä vaakatasoon.



(b) Mittausvalikkojen koko on muuttunut. Oikeaan laitaan on tullut rako mobiililaitteen kääntämisen jälkeen.

Kuva 27: Mittausvalikko kuvattu Sonyn Z1 compactilla.



Kuva 28: Mittausvalikko mobiililaitteen ollessa vaakatasossa.

7 Pohdinta

Tässä työssä on parannettu ja muokattu käyttöjärjestelmän koodia sopivammaksi eri alustoille (erikokoiset tabletit, älypuhelimet). Lisäksi mittausten hoitavaa python koodia on yksinkertaistettu ja siistitty, jotta koodin jatkokehitys ja ylläpito olisi mahdollisimman helppoa. Lisäksi koodiin on lisätty painon mittaus ja sitä kautta painoindeksin mittaus, joka on myös tärkeä terveyteen liittyvä parametri. EKG:tä varten on lisätty eri kanavia tulevaisuutta ajatellen, kun kioskiin saadaan toimivat elektrodit. Tämä diplomityö on vahvistanut, että yksinkertaisen terveyskioskin rakentaminen on erittäin yksinkertaista ja halpaa. Nämä ovat hyvät lähtökohdat kaupallisesta näkökulmasta. Alla on listattu kioskissa käytettyhen komponenttien hinta. Listasta nähdään että kioskin voi hyvin rakentaa noin 600 eurolla.

- HX711 ad-muunnin, 8.95 euroa
- Arduino Yun, 52 euroa
- 3*QL-11A painesensori, ei hintaa saatavilla
- Materiaalikustannukset ei saatavilla
- PM6750-moduuli, 360 euroa

On vielä tehtävää, jotta Aalto Health factoryn Terttu-terveystuoleista saataisiin kaikki niihin liittyvä potentiaali irti. Tätä työtä on ollut erittäin mielenkiintoista tehdä ja opiskella uusia asioita käyttöliittymän ja palvelimen kommunikaatiosta.

Turvallisuudesta, etenkin IoT:n kehittyessä erinäiset tahot ovat kiinnostuneita niiden tallentamien tietojen hyödyntämisestä [8]. Tietoturvan täytyy olla kunnolla suunniteltu siinä vaiheessa, kun kioskia aloitetaan asentamaan julkisiin tiloihin. Pythonille on saatavilla *ssl* ja *smtplib* kirjastot, joilla palvelimelle lähetetyt soketit voidaan salata. Internetistä on myös saatavilla ilmaisia ssl-sertifikaatteja palvelimelle esimerkiksi https://letsencrypt.org/.

Kioskin seuraavaa versiota varten olisi hyvä, jos sen rungon suunnitteluun osallistuisi myös ihmisiä Aalto Health Factorylta. Näin saataisiin mahdollisesti paras hyöty irti niin laitteiden asennuksen kuin kioskin liikuteltavuuden ja erilaisten säätöominaisuuksien kannalta. On myös panostettava enemmän resursseja EKG-elektrodien suunnitteluun, jotta käyttäjät voisivat tulevaisuudessa hyödyntää kioskin koko potentiaalia.

Olisi myös suotavaa, että kioskissa olisi erikokoisia mansetteja eri käsille. Ideaalitilannehan olisi sellainen, missä mansetin sijaan verenpainemittauksen hoitaisi reaaliajassa jokin anturi. Tällainen anturiratkaisu olisi suotuisampi myös siitä näkökulmasta, että käyttäjä ei mahdollisesti jännittäisi mittaustilannetta.

Kioskille ilmaantuu tulevaisuudessa myös haastajia. Tekniikan kehittyessä elintoimintoja mittaavia komponentteja on mahdollista saada pienempään tilaan. Tällainen on esimerkiksi suomalainen, oulussa suunniteltu Oura-hyvinvointisormus [3]. Tällainen teknologia on kuitenkin vielä verrattaen kallista, yksi Oura-sormus maksaa noin 330 euroa. Koska kioskin rakennuskustannukset ovat paljon alhaisemmat

ja sen käyttö ei maksa käyttäjille mitään niin tällä hetkellä kioskeille olisi erittäin hyvät markkinanäkymät.

8 Lähdekoodi

8.1 Arduinon skripti

```
/*
 * This script starts mittaus.py python script on the linux side. Since mittaus.py requires the address
 * node server it must be set in nodeServerAddress.
  This script measures weight as a tuple using HX711.h library and then sends it to linino using
  bridge
  This script also includes printing board's wifi info for convenience
*/

// Arduino weight
#include "HX711.h"
//Bridge Library
#include <Bridge.h>
#include <stdio.h>


//for wifi
#include <Process.h>

// HX711.DOUT  - pin #D7
// HX711.PD_SCK - pin #D8

HX711 scale(7, 8);    // Alustetaan HX711.h kirjastoa varten vaaka: DOUT = digital 7 & SCK = digital 8

//char array is for sending over bridge and long is for measuring
union weightUnion{
  char weightCharArray[10];
  long weightLong;
};

weightUnion weightunion;
boolean isWifiCheck = true;   // change to FALSE to skip wifi
boolean runLinuxScript = true;   //change to FALSE if you dont want the python script to run
Process linuxScript;

void setup() {

  Serial.begin(9600);
  while (!Serial);
  Serial.println("Starting bridge...\n");
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
  Bridge.begin();  // make contact with the linux processor
  digitalWrite(13, HIGH);  // Led on pin 13 turns on when the bridge is ready

```

```
  delay(2000);  // wait 2 seconds

  memset(weightunion.weightCharArray, 0, 10); //zero memory for bridge
//  long readvalue = scale.read_average(10);
//  Serial.print("offset weight: ");
//  Serial.println(readvalue, DEC);

  //scale.tare(10);
  scale.set_offset(8941754);     //platform dry weight, NO CHAIR
  scale.set_scale(14840);        //multiplier to kg
  Serial.println("Scale set up");
}


void loop() {
  if(isWifiCheck){
    Process wifiCheck;
    wifiCheck.runShellCommand("/usr/bin/pretty-wifi-info.lua");  //
    command you want to run

      // while there's any characters coming back from the
      // process, print them to the serial monitor:
      while (wifiCheck.available() > 0) {
        char c = wifiCheck.read();
        Serial.print(c);
      }

    Serial.println();
    isWifiCheck = false;
  }
   if(runLinuxScript) {
    //next command starts the python script server ip address must be
    given
    linuxScript.runShellCommandAsynchronously("python /root/pyscripts/
    mittaus_v5.py -s 10.100.7.124");  //start python script
     while (linuxScript.running()) { //while the script is running send
    weight data to the script
      weightunion.weightLong = scale.get_units(1);
      if(weightunion.weightLong < 0) {    //in case rounding causes it
    to go negative
        weightunion.weightLong = 0;
      }
        Serial.println(weightunion.weightLong, DEC);
        Bridge.put("userWeight", weightunion.weightCharArray);  //
    weight char array to send over to linino
        delay(500);
     }
      runLinuxScript = false;
  }


}
```

Listing 1: sendWeightToLinino.ino

8.2 Python skripti

```
#!/usr/bin/python

#This script needs to be ran from Arduino Yun,
#It handles the data from the scale and PM6750 module and sends it to the server
#This script doesnt check packages received from PM6750 for errors yet

import serial
import sys
import time
import socket
import simplejson as json #simplejson module used since it's more efficient than json, but seems to work identically otherwise
import sys, getopt #passing arguments
#add bridgelibrary to path and import, used at communicating with linino
sys.path.insert(0,'/usr/lib/python2.7/bridge')
from bridgeclient import BridgeClient as bridgeclient

def main(argv):
    SERVER_IP = '' #this must be given as an argument
    SERVER_PORT = 6000
    try:
        opts, args = getopt.getopt(argv,"hs:",["server="])
    except getopt.GetoptError:
        print ("mittaus.py -s|--server <serveraddress>")
        sys.exit(2)
    for opt, arg in opts:
        if opt == '-h':
            print("mittaus.py -s|--server <serveraddress>")
            sys.exit()
        elif opt in ("-s","--server"):
            SERVER_IP = arg

#connect to server
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #create socket object using socket library
        sock.connect((SERVER_IP, SERVER_PORT))
        sock.setblocking(0)
        print("successfully connected to %s" %(SERVER_IP))
    except socket.error as err:
        print("socket creation failed with error: %s" %(err))
        sys.exit(2)

    #Scale configuration
    value = bridgeclient()

    # configure PM6750 serial connection
    serPort = serial.Serial(
        port='/dev/ttyUSB0', #port location in yun's file system
        baudrate=115200, #according to pm6750 specs
```

```
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS,
    # timeout=0.1,
                )
    #Command parameters -> PM6750

    commandDisSpo2 = b"\x55\xaa\x04\x03\x00\xf8" #disable spo2
    commandEnSpo2 = b"\x55\xaa\x04\x03\x01\xf7" #enable spo2
    #commandSetNIBPAdult = b"\x55\xaa\x04\x09\x01\xf1" #set NIBP to
    adult pressure mode
    commandEcgOn = b"\x55\xaa\x04\x01\x01\xf9" #enable ECG output
    commandEcgOff  =b"\x55\xaa\x04\x01\x00\xfa" #disable ECG
    commandStartNIBP = b"\x55\xaa\x04\x02\x01\xf8" #prepare 'start NIBP
    ' command
    commandStopNIBP = b"\x55\xaa\x04\x02\x00\xf9" #stop NIBP
    commandEcgWaveOn = b"\x55\xaa\x04\xfb\x01\xff" #ecg wave on
    commanSpo2WaveOn = b"\x55\xaa\x04\xfe\x01\xfc" #spo2 wave on


    commandEcgLead5 = b"\x55\xaa\x04\x05\x01\xf5" # prepare following
    command ' set ECG to sample lead I (RA-LA, driving by LL) '
    commandEcgLeadI = b"\x55\xaa\x04\x05\x02\xf4" # prepare following
    command ' set ECG to sample lead II (RA-LL, driving by LA) '
    commandEcgLeadII = b"\x55\xaa\x04\x05\x03\xf3" # prepare following
    command ' set ECG to sample lead III (LA-LL, driving by RA) '
    commandEcgLeadIII = b"\x55\xaa\x04\x05\x04\xf2" # default mode,
    lead 5

    commandSoftVerNumber = b"\x55\xaa\x04\xFC\x00\xFF"#Software version
     inquiry
    commandHardVerNumber = b"\x55\xaa\x04\xFD\x00\xFE"#Hardware version
     inquiry


    #These commands set ecg gain
    commandEcgWaveGainx025 = b"\x55\xaa\x04\x07\x01\xf3"
    commandEcgWaveGainx05 = b"\x55\xaa\x04\x07\x02\xf2"
    commandEcgWaveGainx1 = b"\x55\xaa\x04\x07\x03\xf1"
    commandEcgWaveGainx2 = b"\x55\xaa\x04\x07\x04\xf0"

    #These commands set ecg filter mode
    commandEcgFilterModeOper = b"\x55\xaa\x04\x08\x01\xf2"
    commandEcgFilterModeMonitor = b"\x55\xaa\x04\x08\x02\xf1"
    commandEcgFilterModeDiagnose = b"\x55\xaa\x04\x08\x03\xf0"

    hexTail = []              #for initialisation
    MEASUREMENT_ON = False
    WEIGHT_MEASURED = False

    #This function appends rest of the parameters to data that is then
    sent to the server
    def createJSONfromPackage(dataLength, parsedPackFromBerry):
        param = ","
```

```
94          for j in range(dataLength-1):
                param += '"a%d": %d,' %(j+2, int(parsedPackFromBerry[j
    +2],0))
96      #   print(param)
            param = param[:-1]      #this removes the last comma
98          param += ' }'           #adds the ending parenthesis
        # print "final param: %s" %(param)
100         return param

102     #gets weight from arduino, if empty string -> no load
        def measure_Weight():
104         wChar = value.get("userWeight")
            if not wChar:
106             weight = 0
            else:
108             weight = ord(wChar)
            dataw = '{ "tp":"Weight", "a2": %d}' %(weight)
110         sock.setblocking(1)
            sock.settimeout(0.1)
112         #send weight to server
            try:
114             sock.send(dataw)
            except Exception:
116             print("socket timeout")

118     # open serial connection and flushes i/o buffer
        try:
120         serPort.open()
            serPort.flushInput()
122         serPort.flushOutput()
            print "serial port configured"
124     except Exception, e:
            print "error open serial port: " + str(e)
126         exit()


128


130     time.sleep(0.5)

132     #####Main program starts here########

134     while True:
            if (MEASUREMENT_ON):
136             if not WEIGHT_MEASURED:         #only measures weight once
                    measure_Weight()
138                 WEIGHT_MEASURED = True
                bytesInWaiting = serPort.inWaiting()    #check size of
    incoming serial buffer
140     #   print bytesInWaiting
                if(bytesInWaiting >200):        #for performance reasons we
    dont want too big or too small data chunks, 200 seems to work ok
142                 chunk=serPort.read(size=200)      #read a 200 byte
    chuck from serial buffer for further processing. Packages might be
    severed, thereby forming 'tails' which need to be rejoined later
```

```
                hexArray = [hex (ord (char) ) for char in chunk]  # convert every character in chunk to unicode value, then to hex value, and save to hexArray
144             hexArray = hexTail + hexArray   #append the tail end of previous hex array to a new one so it's not lost
                hexTail = []        #clean up the storage variable
146 #       print("hexarray: %s" %(hexArray))
                sock.setblocking(1) #to prevent errno crash (tcp server busy), remember to set sock.setblocking(0) before trying to read socket
148             sock.settimeout(0.1)
                for i in range(0, len(hexArray)-1):
150                 if ((hexArray[i] == "0x55") & (hexArray[i+1] == "0xaa")): #only do stuff if package header 0x55 0xAA is detected, if not, then loop i until it's found
                        #print("hexarray: %s" %(hexArray))
152                     if i+1 >= len(hexArray) - 1 or i  + 2 + int(hexArray[i+2],0) >= len(hexArray)-1:    #if headers + packlength are the last variables in the array or remaingin hexarray doesnt contain enough data as datalength byte indicates
                            hexTail = hexArray[i:len(hexArray)]     # save a the rest of the package to a temp storage variable
154                         break             # processing of current hex array has finished, proceed with data collection

156 #Commands below will read the bit following package headers, that is, the 3rd bit. These packages are coming from PM6750
158 #Example package (nibp):
158 #headers    n   type  A2   A3   A4   A5   A6   checksum
160 #0x55 0xAA 0x08 0x03 0x00 0x00 0x00 0x00 0x00 0xF4
160 # 0     1   2    3    4    5    6    7    8    9

162 #Data package to be sent to the socket is constructed as follows
162 #data = '{ "tp":"packagetype", "vl1": %d ...}', where tp indicates type and vln indicates parameters
164
164 #i+2: skip headers, int(hexArray[i+2],0): pack length excludes the last index
166                 parsedPackFromBerry = hexArray[i+2:i+2 + int(hexArray[i+2],0)-1]

168 #Add rest of the package array to hexTail
                    hexTail = hexArray[i+2 + int(hexArray[i+2],0):-1]
170
                    #that we are not trying to index empty array
172                 if not parsedPackFromBerry:
                        break
174                 #parsedPackFromBerry[i], i=0: length, i=1: datatype, i=2..n-2: data, i=n-1: checksum
                    # if packlength is < 3 discard and start again, invalid length
176                 if(int(parsedPackFromBerry[0],0) < 3):
                        break
```

```
178                         data = "" #init data
                        # print"parsedpack: %s"  %(parsedPackFromBerry)
180                         if (parsedPackFromBerry[1] == "0x1"):       #
    ECG wave
                                dataLength = int(parsedPackFromBerry[0],0)
    - 2
182                             data = '{ "tp":"ECGWave"'
                                param = createJSONfromPackage(dataLength,
    parsedPackFromBerry)
184                             data = data + param
                            # print'parsed ecg wave {}'.format(data)
186                         if (parsedPackFromBerry[1] == '0x2'):   #ECG
    param
                                dataLength = int(parsedPackFromBerry[0],0)
    - 2
188                             data = '{ "tp":"ECGPara"'
                                param = createJSONfromPackage(dataLength,
    parsedPackFromBerry)
190                             data = data + param
                            # print 'parsed hex string {}'.format(data)
192
                            elif (parsedPackFromBerry[1] == "0x3"):       #
    NIBP params: A3=cuff preasure A4=systolic A5=mean A6=dia
194                             dataLength = int(parsedPackFromBerry[0],0)
    - 2
                                data = '{ "tp":"NiBp"'
196                             param = createJSONfromPackage(dataLength,
    parsedPackFromBerry)
                                data = data + param
198
                            elif (parsedPackFromBerry[1] == "0x4"):       #
    SpO2 param
200                             dataLength = int(parsedPackFromBerry[0],0)
    - 2  #True package length
                                #print("datalength: %d" %(dataLength))
202                             data = '{ "tp":"SpO2"'
                                param = createJSONfromPackage(dataLength,
    parsedPackFromBerry)
204                             data = data + param

206                         elif (parsedPackFromBerry[1] == "0xfe"):
                                dataLength = int(parsedPackFromBerry[0],0)
    - 2
208                             data = '{ "tp":"SpO2Wave"'
                                param = createJSONfromPackage(dataLength,
    parsedPackFromBerry)
210                             data = data + param


212
                            elif (parsedPackFromBerry[1] == "0xfc"):       #
    Soft Version A2-A8 ascii bytes represent version
214                             dataLength = int(parsedPackFromBerry[0],0)
    - 2  #True package length
```

```
                        data = '{ "tp":"SoftVer"'       #construct the message
                        param = createJSONfromPackage(dataLength, parsedPackFromBerry)
                        data = data + param


                # print("Data package to sent: %s" %(data))
                    try:
                        if data:    #only send when data contains something
                            sock.send(data)
                        else:
                            continue
                    except Exception:
                        print("socket timeout")
                        break       #stop trying to send data to timed out socket

                    else:   #if ((hexArray[i] == '0x55') & (hexArray[i+1] == '0xaa')):
                        continue


 #!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! 'WHILE MEASUREMENT_ON' loop stops here !!!!!!!!!!!!!!!!!!!!!!!!!


    try:  #this runs regardless of MEASUREMENT_ON
        jsonObject = ""
        sock.setblocking(0) #to allow non blocking socket receive
        a = sock.recv(1) #try reading socket buffer. this will fail if no data in socket buffer, in that case moves to Exception
        while (a != '\n'):  #copies incoming data to a string until termination \n is detected
            jsonObject += a
            a = sock.recv(1)
        jsonMessage = json.loads(jsonObject)  #makes json object from string
    # print 'Read jsonMessage from socket {}'.format(jsonMessage)
 #these read what's in the json
        if jsonMessage['type'] == "startMeasure":
            serPort.flushInput()
            MEASUREMENT_ON = True #enables reading and handling of pm6750 data
            WEIGHT_MEASURED = False
            print "Start measuring"
            serPort.write(commandEnSpo2)
            serPort.write(commandEcgOn)
            serPort.write(commandEcgWaveOn)
            serPort.write(commandEcgLeadII) #enable leadII as default
        if jsonMessage['type'] == "nibpON":
            if (MEASUREMENT_ON):
```

```
                    print "Measuring NIBP"
                    serPort.write(commandStartNIBP) # this sends blood pressure start command signal to pm6750
            if jsonMessage['type'] == "stopMeasure":
                MEASUREMENT_ON = False  #disables reading and handling of pm6750 data
                print "Stopped all measurings"
            if jsonMessage['type'] == "ecg_gainx025":
                serPort.write(commandEcgWaveGainx025)
            if jsonMessage['type'] == "ecg_gainx05":
                serPort.write(commandEcgWaveGainx05)
            if jsonMessage['type'] == "ecg_gainx1":
                serPort.write(commandEcgWaveGainx1)
            if jsonMessage['type'] == "ecg_gainx2":
                serPort.write(commandEcgWaveGainx2)
            if jsonMessage['type'] == "ecg_filterOper":
                serPort.write(commandEcgFilterModeOper)
            if jsonMessage['type'] == "ecg_filterMonitor":
                serPort.write(commandEcgFilterModeMonitor)
            if jsonMessage['type'] == "ecg_filteDiagnose":
                serPort.write(commandEcgFilterModeDiagnose)
            if jsonMessage['type'] == "ecg_setLeadI":
                serPort.write(commandEcgLeadI)
            if jsonMessage['type'] == "ecg_setLeadII":
                serPort.write(commandEcgLeadII)
            if jsonMessage['type'] == "ecg_setLeadIII":
                serPort.write(commandEcgLeadIII)
            if jsonMessage['type'] == "ecg_setLead5":
                serPort.write(commandEcgLead5)


        except Exception: # if no incoming socket data available
            pass
        serPort.flushOutput() #flushes serial outgoing buffer just in case

if __name__=="__main__":
    main(sys.argv[1:])
```

Listing 2: mittaus.py

8.3 Serverin koodi

```
var express = require('express'); //include express module
var app = express();      //invoking express as function returns an express server instance
var http = require('http').Server(app)    //enables use of webSockets with express
var io   = require('socket.io').listen(http); //listens to websocket connections
var net  = require('net');  //TCP-sockets
var fs = require('fs');   //file read/write
```

```
var chairClients = [];
var webClients = [];
var mittaus = 0;

//serves the HTML-part of the web app when user sends GET request
app.get('/', function(req,res){
  res.sendFile(__dirname+'/index.html');
});

/*
app.get('/mittaus', function(req,res){
  currentID = req.param('id');
  res.sendFile(__dirname+'/index.html');
});*/

//get requests for different chairs!
//main page showing all chairs

app.get('/jquery-2.1.3.js', function(req,res){  //serves jquery when requested
  res.sendFile(__dirname+'/jquery-2.1.3.js');
});
/*
app.get('/socket.io/socket.io.js', function(req,res){
  res.sendFile(__dirname+'/socket.io/socket.io.js');
});*/


//var buffer;

// TCP-palvelin joka odottaa yhteyksia tuoleilta// TCP-server
// Jokaisella uudella yhteydenotolla luodaan uusia soketteja// New socket created at every connection
const server = net.createServer(function(socket){


  console.log("Chair is connected");
  var id = 0;
  while (chairClients[id] != null)
    id++;
  chairClients[id] = socket;


  var heartBeat = setInterval(function (){  // pings client regularly to determine if connection is alive
    var messageToChair = {type: 'ping' };


    if (chairClients[id]) {
      chairClients[id].write( JSON.stringify(messageToChair) + '\n');
    }
  }, 5000);
```

```
58  var buffer = '';  //string buffer for incoming data, if problems
      arise, remove "var" and uncomment global "var buffer"

60
    socket.on('data', function(data) {
62

64    buffer += data.toString();
      var newJson;
66    var jsonsProcessed = 0;

68
      while ((i = buffer.indexOf('}')) >= 0) {
70      if (i==0) {
          buffer = buffer.slice(1);
72        continue;
        }
74      newJson = buffer.substring(0,i+1);
      //  console.log(newJson);
76      buffer = buffer.substring(i+1);
        jsonsProcessed +=1;
78      var newJsonText = JSON.parse(newJson);
      //  console.log(newJsonText);
80      if (webClients[id]) {
          webClients[id].emit('dataEvent', newJsonText);
82      }
/*
84      if (mittaus) {
          fs.appendFile('mittauslog.txt', newJson, function (err) {
86          if(err) throw err;
          });
88      }*/
      }
90
//    soketti_web.emit('livedata', { livedata: data });
92//    console.log(JSON.parse(data));

94        });


96
    socket.on('end', function(){
98//    tuoli_sockets.pop();

100   chairClients[id] = null;

102   console.log("Connection with the chair id: "+id+" ,has ended");
    });
104
    socket.on("error", function(){
106
      if (webClients[id]) {
108     webClients[id].emit('socket_error');
      }
110   chairClients[id] = null;
```

```
    socket.destroy();
    console.log("TCP error, connection with the chair id: "+id+" has been terminated");
  });


});
server.listen(6000);  //port number for TCP-connections from chair


io.sockets.on('connection', function(soketti_web){  //websocket (from web app) handling happens here
  mittaus = 0;
  var simulator; //object to be used by interval function

  console.log("Connection from Web-application");

  soketti_web.on('check_chairs', function () {  //happens once when session starts, determines available chairs
    console.log("checking chairs");
    var listOfIDs = [];
    for (i=0; i<100; i++) { // checks first 100 entries in chair and user list
      if ( (chairClients[i] != null) && (webClients[i] == null) )
        listOfIDs.push(i);
    }
    if (listOfIDs.length > 0) {
      soketti_web.emit('show_chairs', listOfIDs);
    }
    else {
      soketti_web.emit('no_chairs');
    }

  });

  soketti_web.on('start', function (id){  //when user clicks "start measure" on web app
    webClients[id]=soketti_web;
    var messageToChair = {type: 'startMeasure' };

    if (chairClients[id]) {
      chairClients[id].write( JSON.stringify(messageToChair) + '\n');
      console.log('Start measuring on id: ' +id+ " with message: " + messageToChair['type']);
    }

    if(id == -1) {  // -1 means user requests simulation
      console.log('Simuloi');
      var j = 0;

      simulator = setInterval(function (){  // simulates continuous measurements
        var messageToChair = {tp: 'ECGWave', a2: Math.floor(Math.random() * 255),
```

```
                    a3: Math.floor(Math.random() * 255),
158                 a4: Math.floor(Math.random() * 255),
                    a5: Math.floor(Math.random() * 255),
160                 a6: Math.floor(Math.random() * 255),
                    a7: Math.floor(Math.random() * 255),
162                 a8: Math.floor(Math.random() * 255)}; //to get random
    integer [0,255] for a2..8 leads
      //  console.log(messageToChair);
164       soketti_web.emit('dataEvent', messageToChair);
          soketti_web.emit('dataEvent', {tp: 'Weight', a2: 95});
166       var messageToChair2 = {tp: 'SpO2Wave', a2: Math.floor(Math.
    random() * 100)}; //to get random integer [0,100]
          soketti_web.emit('dataEvent', messageToChair2);
168       if (j==250) {
            messageToChair = {tp: 'SpO2', a3: (97+Math.floor(Math.random
    () * 2)), a4: (70+Math.floor(Math.random() * 10)), a2: 0};
170         soketti_web.emit('dataEvent', messageToChair);
          //  var ecparam = {tp: 'ECGPara', a2: Math.floor(Math.random()
    * 255)};
172       //  soketti_web.emit('dataEvent', ecparam);

174         j = 0;
          }
176       j++;
        }, 100);
178   }


180
  });
182
  soketti_web.on('stop', function (id){ //when user clicks "stop" in
    web app
184   console.log('Lopeta mittaus');
    var messageToChair = {type: 'stopMeasure' };
186
    if (chairClients[id]) {
188   chairClients[id].write( JSON.stringify(messageToChair) + '\n');
    }
190
    if (id) {
192   webClients[id] = null;
    }
194 if (id == -1) { // -1 means simulation is asked to stop
      clearInterval(simulator);
196 }

198 //    fs.writeFile('mittauslog.txt', '', function (err) {
200 //      if (err) throw err;
//    });
  });
202
  soketti_web.on('nibp', function (id){ //when user clicks "measure
    blood pressure" on web app
204   console.log('nipb request from ' +id);
```

```
    var messageToChair = {type: 'nibpON' }
206
208     if (chairClients[id]) {
          chairClients[id].write( JSON.stringify(messageToChair) + '\n');
210     }
212
        if (id == -1) { // -1 means simulated NIBP is requested
214       messageToClient = {tp: 'NiBp', a4: 0, a6: 0, a2: 4};
          soketti_web.emit('dataEvent', messageToClient);
216
          setTimeout(function() {
218         messageToClient = {tp: 'NiBp', a2: 0, a4: 100+Math.floor(Math.
    random() * 50), a6: 60+Math.floor(Math.random() * 40)};
            soketti_web.emit('dataEvent', messageToClient);
220       }, 5000);
        }
222   });

224 /*  soketti_web.on('nibpStop', function (id){ //when user clicks "
    measure blood pressure" on web app
        console.log('stop ');
226     var messageToChair = {type: 'nibpStop' }
228
        if (chairClients[id]) {
230       chairClients[id].write( JSON.stringify(messageToChair) + '\n');
        }
232
234     if (id == -1) { // -1 means simulated NIBP is requested
          messageToClient = {tp: 'NiBp', a4: (0), a6: (0), a2: 8};
236       soketti_web.emit('dataEvent', messageToClient);
238     }
240
    }); */
242   soketti_web.on('lead_select', function(lead, id) {
        console.log("set lead: "+ lead + " chair: " +id);
244     var messageToChair = {type: lead}
        if(chairClients[id]) {
246       chairClients[id].write( JSON.stringify(messageToChair) +'\n');
        }
248     if(id == -1) {
          if (lead == "ecg_setLeadI") {
250         messageToClient = {tp: 'ECGPara', a2: 64};  //send only bit
    string 0100 0000
          }else if (lead == "ecg_setLeadII") {
252         messageToClient = {tp: 'ECGPara', a2: 128}; // 1000 0000
          }else if (lead == "ecg_setLeadIII") {
254         messageToClient = {tp: 'ECGPara', a2: 192}; // 1100 0000
          }else if (lead == "ecg_setLead5") {
```

```
256         messageToClient = {tp: 'ECGPara', a2: 0};
        }
258       soketti_web.emit('dataEvent', messageToClient);
      }
260   });
    soketti_web.on('gain_select', function(gain, id) {
      console.log("set gain: "+ gain + " chair: " +id);
      var messageToChair = {type: gain}
      if(chairClients[id]) {
        chairClients[id].write( JSON.stringify(messageToChair) +'\n');
      }
      if(id == -1) {
        if (gain == "ecg_gainx025") {
          messageToClient = {tp: 'ECGPara', a2: 0}; //send only bit
     string 0000 0000
        }else if (gain == "ecg_gainx05") {
          messageToClient = {tp: 'ECGPara', a2: 4}; // 1000 0000
        }else if (gain == "ecg_gainx1") {
          messageToClient = {tp: 'ECGPara', a2: 8}; // 1100 0000
        }else if (gain == "ecg_gainx2"){
          messageToClient = {tp: 'ECGPara', a2: 12};
        }
        soketti_web.emit('dataEvent', messageToClient);
      }
    });
    soketti_web.on('filtermode_select', function(fmode, id) {
      console.log("set filtermode: "+ fmode + " chair: " +id);
      var messageToChair = {type: fmode}
      if(chairClients[id]) {
        chairClients[id].write( JSON.stringify(messageToChair) +'\n');
      }
      if(id == -1) {
        if (fmode == "ecg_filterOper") {
          messageToClient = {tp: 'ECGPara', a2: 0}; //send only bit
     string 0100 0000
        }else if (fmode == "ecg_filterMonitor") {
          messageToClient = {tp: 'ECGPara', a2: 16};  // 1000 0000
        }else if (fmode == "ecg_filteDiagnose") {
          messageToClient = {tp: 'ECGPara', a2: 32};  // 1100 0000
        }
        soketti_web.emit('dataEvent', messageToClient);
      }
    });
    soketti_web.on('disconnect', function() {   //when user closes the
     web page
      console.log('User disconnection');
      var i = webClients.indexOf(soketti_web);
      if (i >= 0) {
        var messageToChair = {type: 'stopMeasure'};
        if(chairClients[i]) {
          chairClients[i].write( JSON.stringify(messageToChair) + '\n');
        }
        webClients[i] = null;
      }
```

```
    clearInterval(simulator); //this kills the simulator
  });

/*
  soketti_web.on('tallenna', function() {
    date = new Date();
    var now = date.getTime();
    var dateString = now.toString();

    fs.rename('mittauslog.txt', dateString+'.txt', function(err) {
      if(err) throw err;
    });
  }); */



});

/*
http.listen(8080, function(){
  console.log('listening on *:8080');
});
*/
http.listen(8081, "0.0.0.0"); //listens to http requests at this port,
   0.0.0.0 means everyone is served
```

Listing 3: server.js

8.4 Käyttöliittymän koodi

```
<!doctype html>
<html>

<style>

.hidden{
  display:none;
  }
.unhidden{
  display:block;
  }
</style>

<head>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
 <meta name="viewport" content="width=device-width,initial-scale=1,
    maximum-scale=1,user-scalable=no"/>
 <meta name="apple-mobile-web-app-capable" content="yes">
 <meta name="apple-mobile-web-app-status-bar-style" content="black">

 <title>Aalto Health Kiosk User Interface</title>
```

```
22 <link rel="stylesheet" href="http://www.w3schools.com/lib/w3.css">

24 <script> src="https://cdnjs.cloudflare.com/ajax/libs/jquery-mobile
     /1.4.5/jquery.mobile.min.js"</script>
   <script src="socket.io/socket.io.js"></script>
26 <script src="jquery-2.1.3.js"></script>
   <script data-require="flot@*" data-semver="0.0.7" src="//cdnjs.
     cloudflare.com/ajax/libs/flot/0.7/jquery.flot.min.js"></script>
28 <script data-require="flot@*" data-semver="0.0.7" src="//cdnjs.
     cloudflare.com/ajax/libs/flot/0.7/jquery.flot.threshold.min.js"></
     script>
   <script data-require="flot@*" data-semver="0.0.7" src="//cdnjs.
     cloudflare.com/ajax/libs/flot/0.7/jquery.flot.symbol.min.js"></
     script>
30 <script data-require="flot@*" data-semver="0.0.7" src="//cdnjs.
     cloudflare.com/ajax/libs/flot/0.7/jquery.flot.stack.min.js"></
     script>
   <script data-require="flot@*" data-semver="0.0.7" src="//cdnjs.
     cloudflare.com/ajax/libs/flot/0.7/jquery.flot.selection.min.js"></
     script>
32 <script data-require="flot@*" data-semver="0.0.7" src="//cdnjs.
     cloudflare.com/ajax/libs/flot/0.7/jquery.flot.resize.min.js"></
     script>
   <script data-require="flot@*" data-semver="0.0.7" src="//cdnjs.
     cloudflare.com/ajax/libs/flot/0.7/jquery.colorhelpers.min.js"></
     script>
34 <script data-require="flot@*" data-semver="0.0.7" src="//cdnjs.
     cloudflare.com/ajax/libs/flot/0.7/jquery.flot.image.min.js"></
     script>
   <script data-require="flot@*" data-semver="0.0.7" src="//cdnjs.
     cloudflare.com/ajax/libs/flot/0.7/jquery.flot.fillbetween.min.js"
     ></script>
36 <script data-require="flot@*" data-semver="0.0.7" src="//cdnjs.
     cloudflare.com/ajax/libs/flot/0.7/jquery.flot.crosshair.min.js"></
     script>
   <script data-require="flot@*" data-semver="0.0.7" src="//cdnjs.
     cloudflare.com/ajax/libs/flot/0.7/excanvas.min.js"></script>
38 <script data-require="flot@*" data-semver="0.0.7" src="//cdnjs.
     cloudflare.com/ajax/libs/flot/0.7/jquery.flot.navigate.min.js"></
     script>

40 </head>

42 <script>

44 var socket   = io.connect();
   var id;

46
   $(document).ready(function(){

48   var socket   = io.connect();
50   var id;

52   socket.emit('check_chairs');
```

```
54  socket.on('show_chairs', function(listOfIDs) {
      console.log("available chairs");
56  var btnav;
      if (listOfIDs != null) {
58      for (var i=0; i<listOfIDs.length; i++) {
          if (listOfIDs[i] != null) {
60          var buttons="";
            //buttons += '<h2>Available chairs:</h2> <input type="button"
    id = "button_'+listOfIDs[i]+'" class="selectChair" value="'+
   listOfIDs[i]+'"></input>';
62          btnav = $("<button></button>").text(listOfIDs[i]);
            btnav.prop('id',"button_"+listOfIDs[i]);
64                          btnav.prop('class',"selectChair");
                        btnav.prop('value',listOfIDs[i]);
66          console.log(btnav);
          }
68      }
        $('#chair_menu').append(btnav);
70      $('#chair_status').text("Available chairs");
        //document.getElementById("chair_menu").innerHTML = buttons;
72    }
  });
74
  socket.on('no_chairs', function() {
76        console.log("no chairs found");
    $('#chair_status').text("No available chairs");
78  });


80
  $('#chair_menu').on('click', '.selectChair', function() {
82    id = this.value;
    console.log(id);
84    document.getElementById("menu").className = 'hidden';
    document.getElementById("results").className = 'unhidden';
86        $('#userchair_id').text(id);
  });
88
  $('#simulaattori').click(function(){
90    id = -1;
    console.log(id);
92    document.getElementById("menu").className = 'hidden';
    document.getElementById("results").className = 'unhidden';
94        $('#userchair_id').text(id+" (simulate)");
        });
96
        $('#startMeasure').click(function(){
98    socket.emit('start', id);
        });
100 $('#stopMeasure').click(function() {
    socket.emit('stop', id);
102 });

104 $('#measureNIBP').click(function() {
```

```
    socket.emit('nibp', id);
});
$('#saveData').click(function() {
    socket.emit('tallenna', id);
});
$('#backToChairs').click(function(){
    socket.emit('stop', id);
    location.reload(true);
});
$("input[name=lead_select]").change(function(){
if($("input[name=lead_select]").is(':checked'))
    {
    $ans = $("input[name=lead_select]:checked").val();
    $(':radio').prop('checked');
            socket.emit('lead_select',$ans,id);
    }
});
$("input[name=gain_select]").change(function(){
if($("input[name=gain_select]").is(':checked'))
    {
    $ans = $("input[name=gain_select]:checked").val();
    $(':radio').prop('checked');
            socket.emit('gain_select',$ans,id);
    }
});
$("input[name=filtermode_select]").change(function(){
if($("input[name=filtermode_select]").is(':checked'))
    {
    $ans = $("input[name=filtermode_select]:checked").val();
    $(':radio').prop('checked');
            socket.emit('filtermode_select',$ans,id);
    }
});

$("input[name=userheight]").on("change keyup", function(){
    var weight = parseInt($('output[name=weight_status]').val());
    var status = "";
    //console.log("weight: " + weight);
    if(weight) {
      var height = $('input[name=userheight]').val();
    //  console.log("height: " +height);
      if (height > 0){
        var bmi = weight/(height*height)*10000;
        bmi = bmi.toFixed(1);
      //  console.log(bmi);
        if(bmi < 18.5) {
          status = "Underweight";
          $('#weightstatus').css({"color":"#ff0000"});
        }else if(bmi < 25){
          status = "Normal weight";
          $('#weightstatus').css({"color":"green"});

        }else if(bmi < 30){
          status = "Overweight";
```

```
            $('#weightstatus').css("color","#ff7f7f");
160         }else{
              status ="Obesity";
162           $('#weightstatus').css({"color":"#ff0000"});
            }
164         $('output[name=bmi]').val(bmi);
            $('#weightstatus').text(status)
166       }
      }
168   });

170   $('#placeholder_ecg').bind("plothover", function (event, pos, item) {
      if(item){
172     $('#placeholder_ecglabel').text(item.series.label);
      } else {
174     $('#placeholder_ecglabel').text("");
      }
176
      });
178
      var i,
180   now = new Date();
      var ecg1_data = [];
182   var ecg2_data = [];
      var ecg3_data = [];
184   var ecg4_data = [];
        var ecg5_data = [];
186   var ecg6_data = [];
      var ecg7_data = [];
188
        var data_spo2 = [];
190     totalPoints = 1000;

192 // dataset to print ecg wave
    dataset = function(){
194     return [
        {
196         label: "ECG Lead I",
            data: ecg1_data
198     },
        {
200         label: "ECG Lead II",
            data: ecg2_data
202     },
        {
204         label: "ECG Lead III",
            data: ecg3_data
206     },
        {
208         label: "ECG Lead aVR",
            data: ecg4_data
210     },
        {
212         label: "ECG Lead aVL",
```

```
        data: ecg5_data,
    },
    {
        label: "ECG Lead aVF",
        data: ecg6_data
    },
    {
        label: "ECG Lead V",
        data: ecg7_data
    }
];
}



//populates empty graph
      for (i=0; i<totalPoints;i++){
    now.setMilliseconds(now.getMilliseconds() + 1)
    ecg1_data.push([now.getTime(), 0]);
    ecg2_data.push([now.getTime(), 0]);
    ecg3_data.push([now.getTime(), 0]);
    ecg4_data.push([now.getTime(), 0]);
    ecg5_data.push([now.getTime(), 0]);
    ecg6_data.push([now.getTime(), 0]);
    ecg7_data.push([now.getTime(), 0]);
          data_spo2.push([now.getTime(), 0]);
      }

  var plot_ecg = $.plot("#placeholder_ecg", dataset(), {
    series: {
      shadowSize: 0
    },
    yaxis: {
      min: 0,
      max: 1750
    },
    xaxis: {
      //set x-axis as time
      mode: "time",
            timeformat: "%h:%M:%S"
    },
          legend: {
            show: false
//              position: "nw"
              // noColumns: 1
            },
            grid: {
              hoverable: true
            }
  });

  var plot_spwave = $.plot("#placeholder_sp02wave", [data_spo2], {
    series: {
      shadowSize: 0,
```

```
    //  label: "Sp02 Wave Graph"
    },
    yaxis: {
      min: 0,
      max: 100
    },
    xaxis: {
      //set x-axis as time
      mode: "time",
            timeformat: "%h:%M:%S"
    },
/*    legend: {
      container: $("#legend_sp02wave")
    }*/
  });


  function update(){
          plot_ecg.setData(dataset());
          plot_ecg.setupGrid();
          plot_ecg.draw();

          plot_spwave.setData([data_spo2]);
          plot_spwave.setupGrid();
          plot_spwave.draw();

          setTimeout(update, 1000);
  }

    /*Global variables */
  var ecg_leadmode = "Lead V";
  var ecg_signal = "normal";
  var lead_status = "normal";
  var ecg_gain = 0.25;
  var ecg_filtermode = "Operation mode";

  socket.on('dataEvent', function (mittausData) {


//    console.log(mittausData);
//    var json = JSON.parse(mittausData);
    var json = mittausData;
//    console.log(json.a2);
    if(json.tp == 'SpO2') {
      if(json.a2 != '0') {
        $('output[name=spo2status]').val("Searching for signal..");
      }
      else
        $('output[name=spo2status]').val("Measuring..");
        $('output[name=oxy_status]').val(json.a3);
        $('output[name=hr_status]').val(json.a4);
    }

    if(json.tp == 'NiBp') {
```

```
      if ((json.a2 == 0) && (json.a4 != 0)) {
322     $('output[name=nibpstatus]').val("Measured");
        $('output[name=sys_status]').val(json.a4);
324     $('output[name=dia_status]').val(json.a6);
      }
326   else if (json.a2 && 4 == 4) {
        $('output[name=nibpstatus]').val("Measuring..");
328   }
      else if (json.a2  && 16 == 16) {
330     $('output[name=nibpstatus]').val("Please check cuff's
    attachment");
      }
332   else if (json.a2 && 8 == 8) {
        $('output[name=nibpstatus]').val("Measurement stopped");
334   }
    }
336


338 if (json.tp == 'ECGWave') {

340   //input new data in ecg graph
    //  data_ecg.shift();
342
    //  data_ecg.push([now.getTime(), json.a2])
344   ecg1_data.shift();
      ecg2_data.shift();
346   ecg3_data.shift();
      ecg4_data.shift();
348   ecg5_data.shift();
      ecg6_data.shift();
350   ecg7_data.shift();
      now.setMilliseconds(now.getMilliseconds()+4)
352   ecg1_data.push([now.getTime(), json.a2]);
      ecg2_data.push([now.getTime(), json.a3+250]);
354   ecg3_data.push([now.getTime(), json.a4+500]);
      ecg4_data.push([now.getTime(), json.a5+750]);
356   ecg5_data.push([now.getTime(), json.a6+1000]);
      ecg6_data.push([now.getTime(), json.a7+1250]);
358   ecg7_data.push([now.getTime(), json.a8+1500]);


360


362 }
    if (json.tp == 'SpO2Wave') {
364   //input new data in ecg graph
      //document.getElementById(json.tp).innerHTML = 'Measuring Sp02
    Wave...';
366   data_spo2.shift();
      now.setMilliseconds(now.getMilliseconds()+4);
368   data_spo2.push([now.getTime(), json.a2]);
    //  document.getElementById(json.tp).innerHTML = data_spo2;
370 }
    if (json.tp == 'Weight') {
372   $('output[name=weight_status]').val(json.a2);
```

```
    }
374   if (json.tp == 'ECGPara') {
      var base2 = json.a2.toString(2);
376   var paddedBase2 = pad(base2, 8);
/*    var ecg_leadmode = "Lead V";
378   var ecg_signal = "normal";
      var lead_status = "normal";
380   var ecg_gain = 0.25;
      var ecg_filtermode = "Operation mode";
382 */
      console.log(paddedBase2)
384
      if(paddedBase2[7] == 1){  //index is reversed
386     ecg_signal = "weak";
      }else if(paddedBase2[0] == 0){
388     ecg_signal = "normal";
      }
390   if(paddedBase2[6] == 1){
        lead_status = "lead off";
392   }else  if(paddedBase2[1] == 0){
        lead_status = "normal";
394   }

396   if(paddedBase2[4] == 0 && paddedBase2[5] == 0){   // bits[3][2]:
    11 => gain 2
        ecg_gain = 0.25;
398   }else if(paddedBase2[4] == 1 && paddedBase2[5] == 1){   // bits
    [3][2]: 11 => gain 2
        ecg_gain = 2;
400   }else if(paddedBase2[4] == 0 && paddedBase2[5] == 1){       //
    bits: 01 => gain 1
        ecg_gain = 0.5;
402   }else if(paddedBase2[4] == 1 && paddedBase2[5] == 0){       //
    bits: 10 => gain 1
        ecg_gain = 1;
404   }

406   if(paddedBase2[2] == 0 && paddedBase2[3] == 0){   //bits[5][4]
        ecg_filtermode = "Operation mode";
408   }else if(paddedBase2[2] == 0 && paddedBase2[3] == 1){
        ecg_filtermode = "Monitor mode";
410   }else if(paddedBase2[2] == 1 && paddedBase2[3] == 0){
        ecg_filtermode = "Diagnose mode";
412   }

414   if(paddedBase2[0] == 0 && paddedBase2[1] == 0){   //bits[7][6]
        ecg_leadmode = "Lead 5";
416   }else if(paddedBase2[0] == 0 && paddedBase2[1] == 1){
        ecg_leadmode = "Lead I";
418   }else if(paddedBase2[0] == 1 && paddedBase2[1] == 0){
        ecg_leadmode = "Lead II";
420   }else if(paddedBase2[0] == 1 && paddedBase2[1] == 1){
        ecg_leadmode = "Lead III";
422   }
```

```
424
        $('output[name=ecg_signal_status]').val(ecg_signal);
426     $('output[name=ecg_lead_status]').val(lead_status);

428   }
//    console.log(data);
430 });

432 function pad(number, length) {

434   var str = '' + number;
      while (str.length < length) {
436     str = '0' + str;
      }
438
  return str;
440
  }
442 update();

444 });

446 </script>

448 <body>

450 <div id="menu" class="unhidden w3-container w3-section w3-row-padding">
 <div id="simulate" class="w3-container w3-section w3-row-padding">
452  <div class="w3-col l3 s3 m3">
    <button  class="w3-btn-block w3-teal" id="simulaattori">Simulate
     chair</button>
454  </div>
 </div>
456 <div class="w3-container w3-section">
  <h2 id="chair_status"></h2>
458 <div id="chair_menu" class="w3-container w3-section"></div>
 </div>
460 </div>


462
<div id="results" class="hidden w3-container w3-section w3-row-padding"
    >
464
<ul class="w3-navbar w3-border w3-light-grey">
466 <li><button class="w3-btn-block w3-teal w3-left-align" id="
    startMeasure">Start measuring</button>
  <li><button class="w3-btn-block w3-teal w3-left-align" id="
    stopMeasure">Stop measuring</button></li>
468 <li><button class="w3-btn-block w3-teal w3-left-align" id="
    measureNIBP">Measure blood pressure</button></li>
  <li><button class="w3-btn-block w3-teal w3-left-align" id="saveData">
    Save</button></li>
```

```
  <li><button class="w3-btn-block w3-teal w3-left-align" id="
    backToChairs">Back to Menu</button></li>
</ul>

<!--div id="menu" class="w3-container w3-section">
</div-->
<div class="w3-row">
<header class=" w3-container w3-teal w3-col l12 s12">
  <h1>Terttu Health Kiosk: Using chair <span id="userchair_id"></span
    ></h1>
</header>
</div>

<div class="w3-row">
 <div id="NiBp" class="w3-container w3-col l3 s12 m4">
  <p><b>Blood pressure status: </b><output name="nibpstatus">Not
    measured</output></p>
  <p><b>Sys: </b><output name="sys_status" style="margin: 10px;">-</
    output> <b>Dia: </b><output name="dia_status" style="margin: 10px;"
    >-</output></p>
 </div>

 <div class="w3-border-left w3-border-right w3-border-top w3-container
    w3-col l4 s12 m4">
  <p><b>Weight status: </b><output name="weight_status" style="margin:
    10px;">-</output><span>kg</span></p>
  <form >
   <p>(Optional) Enter height in cm to calculate BMI</p>
   <input type="number" name="userheight" value="0" autocomplete='off'
    ></input> = <output name="bmi"></output> 
   <span id="weightstatus"></span>
  </form>
 <br>
 </div>
 <div class=" w3-border-top w3-container w3-col l5 s12 m4">

 </div>
</div>

<div class="w3-container w3-border w3-row">
 <div class="w3-col" style="width:30%">
  <p><b>SpO2 status: </b><output name="spo2status">Not measured</output
    ></p>
  <p><b>Oxygen saturation: </b><output name="oxy_status" style="margin:
     10px;">-</output><span style="padding-right: 1cm">%
  </span></p>
  <p><b>Heart rate: </b><output name="hr_status" style="margin: 10px;"
    >-</output></p>
 </div>

 <div class="w3-col w3-section" id="placeholder_sp02wave" style="width
    :70%;height:250px"></div>
 <!--div class="w3-col w3-container" id="legend_sp02wave" ></div-->
<br>
```

```
512
</div>
514

516   <!--h2 id="SpO2Wave">SpO2Wave not measured</h2-->

518
<div class="w3-container w3-border w3-row">
520  <div class="w3-col" style="width:40%">
   <p><b>ECG signal status: </b><output name="ecg_signal_status">Not
     measured</output></p>
522   <p><b>ECG lead status: </b><output name="ecg_lead_status">Not
     measured</output></p>

524   <form name="lead_selector">
    <label for="lead_selector"><b>Select lead: </b></label><br>
526    <input type="radio" name="lead_select" value="ecg_setLeadI"> Lead I<
     br>
    <input type="radio" name="lead_select" value="ecg_setLeadII" checked
     > Lead II<br>
528    <input type="radio" name="lead_select" value="ecg_setLeadIII"> Lead
     III<br>
    <input type="radio" name="lead_select" value="ecg_setLead5"> Lead V<
     br><br>
530   </form>

532   <form name="gain_selector">
    <label for="gain_selector"><b>Select Gain: </b></label><br>
534    <input type="radio" name="gain_select" value="ecg_gainx025"> Gain x0
     .25<br>
    <input type="radio" name="gain_select" value="ecg_gainx05" checked>
     Gain x0.5<br>
536    <input type="radio" name="gain_select" value="ecg_gainx1"> Gain x1<
     br>
    <input type="radio" name="gain_select" value="ecg_gainx2"> Gain x2<
     br><br>
538   </form>
   <form name="ecg_filtermode">
540    <label for="ecg_filtermode"><b>Select Filter mode: </b></label><br>
    <input type="radio" name="filtermode_select" value="ecg_filterOper">
      Operation mode, 1-25Hz<br>
542    <input type="radio" name="filtermode_select" value="
     ecg_filterMonitor" checked> Monitor mode, 0.5-75Hz<br>
    <input type="radio" name="filtermode_select" value="
     ecg_filteDiagnose"> Diagnose mode, 0.05-100Hz
544   </form>
<br>
546  </div>
  <div class="w3-col w3-section" id="placeholder_ecg" style="width:60%;
     height:400px"></div>
548  <div class="w3-col w3-container" id="placeholder_ecglabel" style="
     width:60%"></div>

550<br>
```

```
</div>
552

</div> <!--Results-->

556 </body>
</html>
```

Listing 4: index.html

Viitteet

[1] Avia Semiconductor. *24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales*. Saatavissa: http://www.dfrobot.com/image/data/SEN0160/hx711_english.pdf Viitattu 4.1.2016.

[2] bogde. An arduino library to interface the avia semiconductor hx711 24-bit analog-to-digital converter (adc) for weight scales., 2016. Saatavissa: https://github.com/bogde/HX711.

[3] Landowski E. Oura-hyvinvointisormus haastaa aktiivisuusrannekkeet. *Tekes*, 11 2015. Saatavissa https://www.tekes.fi/tekes/tulokset-ja-vaikutukset/caset/2015/oura-hyvinvointisormus-haastaa-aktiivisuusrannekkeet/. Viitattu 27.6.2016.

[4] Centers for Disease Control and Prevention. Division of nutrition, physical activity, and obesity. http://www.cdc.gov/healthyweight/assessing/bmi/adult_bmi/index.html, 2015. Viitattu 28.3.2016.

[5] Kyle U. G. Bioelectrical impedance analysis—part i: review of principles and methods. *Clinical Nutrition*, 23:1226–1243, 2004.

[6] Jones M. H. *A Practical Introduction to Electronic Circuits, Third Ed.* Cambridge University Press, 1995.

[7] Makkonen J. Verenpaineen mittaus mems-paineantureilla. Master's thesis, Aalto yliopisto, 2014. Saatavissa: https://aaltodoc.aalto.fi/handle/123456789/12723.

[8] McLaughlin J. Nsa looking to exploit internet of things, including biomedical devices, official says. *The Intercept*, 6 2016. Saatavissa https://theintercept.com/2016/06/10/nsa-looking-to-exploit-internet-of-things-including-biomedical-devices-official Viitattu 13.6.2016.

[9] Santero J-M. Fysiologisia suureita mittaavan hajautetun järjestelmän suunnittelu ja toteutus. Master's thesis, Aalto yliopisto, 2015. Saatavissa: https://aaltodoc.aalto.fi/handle/123456789/19113.

[10] Appel L. J. ja Brands M. W. ja Daniels S. R. ja Karanja N. ja Elmer P. J. ja Sacks F. M. Dietary approaches to prevent and treat hypertension: A scientific statement from the american heart association. *Hypertension*, 47(2):296–308, 2006.

[11] Tortora G. J. ja Derrickson B. *Principles of Anatomy and Physiology 12th Edition*. John Wiley and Sons, Inc, 2009.

[12] Crawford J. ja Doherty L. *Practical Aspects of ECG Recording*. Cumbria, GBR, 2012.

[13] Pickering T. G. ja Hall J. E. ja Appel L. J. ja Falkner B. E. ja Graves. J. ja Hill M. N. ja Jones D. W. ja Kurtz T. ja Sheps S. G. ja Roccella E. J. Recommendations for blood pressure measurement in humans and experimental animals: Part 1: Blood pressure measurement in humans: A statement for professionals from the subcommittee of professional and public education of the american heart association council on high blood pressure research. *Hypertension*, 45(1):142–161, 2005.

[14] Malmivuo J. ja Plonsey R. *Bioelectromagnetism - Principles and Applications of Bioelectric and Biomagnetic Fields*. Oxford University Press, 1995. Saatavissa: http://www.bem.fi/book/index.htm.

[15] Piccoli A. ja Rossi B. ja Pillon L. ja Bucciante G. A new method for monitoring body fluid variation by bioimpedance analysis: The rxc graph. *Kidney International*, 46(2):534–539, 8 1994.

[16] Alba N.A. ja Sclabassi R.J. ja Mingui S. ja Cui X.T. Novel hydrogel-based preparation-free eeg electrode. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 18(4):415–423, Aug 2010.

[17] Joshi A. ja Trout K. The role of health information kiosks in diverse settings: a systematic review. *Health Information and Libraries Journal*, 31(4):254 – 273, 2014.

[18] Lääkärilehti. Terveyskioski vähensi terveyskeskuksen kuormaa, 2011. Saatavissa http://www.laakarilehti.fi/ajassa/ajankohtaista/terveyskioski-vahensi-terveyskeskuksen-kuormaa/. Viitattu 30.5.2016.

[19] Martin D. ja Lawler J. Mengelkoch L. J. A review of the principles of pulse oximetry and accuracy of pulse oximeter estimates during exercise. *Physical Therapy*, 74(1):40–49, 1994.

[20] Young A. J. ja Sawka M. N. O'Brien C. Bioelectrical impedance to estimate changes in hydration status. *Int J Sports Med*, 23(5), 7 2002.

[21] QY Electronic Co Ltd. *QL-11A*. Saaatavissa: http://www.qysensor.com/uploadfile/201310/20131014104350291.pdf Viitattu 4.1.2016.

[22] Haymond S. Oxygen saturation a guide to laboratory assessment. *Clinical Laboratory News*, 2006. Saatavissa: http://www.optimedical.com/pdf/articles/oxygen-saturation-laboratory-assessment.pdf. Viitattu 31.3.2016.

[23] Lopez S. *Pulse Oximeter Fundamentals and Design*. Freescale Semiconductor, 11 2012.

[24] Wunker S. A new age for healthcare kiosks – five ways next generation kiosks disrupt medicine and healthcare marketing. *Forbes*, 2013. Saatavissa: http://www.forbes.com/sites/stephenwunker/2013/07/16/

a-new-age-for-healthcare-kiosks-five-ways-next-generation-kiosks-disrupt-medici
#5bac8dce6d71. Viitattu 23.5.2016.

[25] Shanghai Berry electronic tech co ltd. *Patient Monitor 6750*. Saatavissa: http://www.berry-med.com/UploadFiles/20140820130151314.pdf Viitattu 4.1.2016.

[26] Sitra. Terveyskioski - uudenlainen tapa hoitaa terveyspalveluita, 2012. Saatavissa http://www.sitra.fi/hyvinvointi/terveyskioski. Viitattu 30.5.2016.

[27] Sotera Wireless. *ViSi Mobile wireless patient monitoring system*. Saatavissa: http://www.soterawireless.com/wp-content/uploads/2013/10/new_cNIBP_clearance_PR010-100913Final.pdf Viitattu 17.2.2016.

[28] Taltioni – maksuton ja turvallinen terveystili kaikille suomalaisille. Saatavissa: http://www.taltioni.fi Viitattu 3.3.2016.

[29] United Nations Population Fund (UNFPA), New York, and HelpAge International, London. *Ageing in the Twenty-First Century: A Celebration and A Challenge*, United Nations Population Fund 605 Third Avenue, New York, NY 10158, USA hq@unfpa.org www.unfpa.org, 2012. Saatavissa: http://www.helpage.org/download/50af6e9c8f44b.

[30] Yle. Terveyskioski palvelee hyvin, muttei miehiä ja nuoria, 2011. Saatavissa http://yle.fi/uutiset/terveyskioski_palvelee_hyvin_muttei_miehia_ja_nuoria/2647733. Viitattu 30.5.2016.